Aalto-yliopisto
Sähkötekniikan
korkeakoulu

Ville Saari

Radioastronomisten mittausten häiriötekijöiden monitorointi ohjelmistoradiolla

Diplomityö, joka on jätetty opinnäytteenä
tarkastettavaksi diplomi-insinöörin tutkintoa varten.

Espoossa 19.01.2015
Työn valvoja: Professori Jorma Skyttä
Työn ohjaaja: DI Petri Kirves

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU
PL 13000
00076 AALTO

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Ville Saari
Työn nimi: Radioastronomisten mittausten häiriötekijöiden monitorointi ohjelmistoradiolla
Koulutusohjelma: Signaalinkäsittely
Päiväys: 19.01.2015 Sivumäärä: [74 + 37]
Työn valvoja: Professori Jorma Skyttä
Työn ohjaaja: DI Petri Kirves
Kieli: Suomi

Langattomia yhteyksiä hyödyntävien laitteiden ja sovellusten määrä kasvaa jatkuvasti, mikä lisää myös radioympäristön sisältämien häiriöiden määrää. Laitteiden hyödyntämät yhteydet myös kehittyvät entistä dynaamisempaan suuntaan, mikä monimutkaistaa niistä aiheutuvia häiriöitä. Samalla radioastronomiassa käytetyt entistä herkemmät radiovastaanottimet tekevät mittauksista herkempiä tehotasoltaan yhä pienemmille häiriöille, korostaen häiriöiden monitoroinnin ja suodatuksen tärkeyttä radioastronomian tieteenalalla.

Tämän diplomityön tavoitteena oli toteuttaa ohjaus- ja tiedonkeräysohjelmisto Ettus Research USRP N210 ohjelmistoradiolle radioastronomisten mittausten häiriötekijöiden monitorointiin. Järjestelmällä pyrittiin vastaamaan radioastronomisten mittausten kasvavan häiriöherkkyyden sekä häiriöympäristön kehityksen asettamiin haasteisiin. Ohjelmiston toteuttamisen lisäksi työssä tutkittiin ja karakterisoitiin USRP N210 -ohjelmistoradiojärjestelmän radioteknisiä ominaisuuksia, kuten laitteen herkkyyttä, pyyhkäisynopeutta, taajuusvastetta sekä laitteen kaistanleveyden selektiivisyyttä. Työssä myös selvitettiin järjestelmän suorituskykyä rajoittavia tekijöitä, millä pyrittiin tehostamaan ja edesauttamaan järjestelmän jatkokehittämistä.

Työssä toteutetulla ohjelmistolla voidaan tukea Metsähovin nykyisen häiriömonitorointijärjestelmän toimintaa. Työssä hyödynnetyn ohjelmistoradiolaitteiston avulla voidaan esimerkiksi saavuttaa Metsähovin nykyistä häiriömonitorointijärjestelmää huomattavasti parempi aikatason resoluutio. Lisäksi toteutetulla järjestelmällä saavutetaan suurempi herkkyys nykyistä häiriömonitorointijärjestelmää pienemmillä pyyhkäisyajoilla. Järjestelmä mahdollistaa siis entistä lyhyempien ja tehotasoltaan pienempien radiohäiriöiden havaitsemisen.

Luotettavien absoluuttisten mittausten suorittaminen USRP N210 -ohjelmistoradiojärjestelmällä vaatii kuitenkin laitteen kalibrointia sekä laitteen ulkoisten- ja sisäisten virhelähteiden huomiointia. Laitteen kalibrointiin ja mittaustulosten suodattamiseen kiinnitettiinkin erityistä huomiota laitteistoa ohjaavaa ohjelmistoa kehitettäessä. Järjestelmän mittaustulosten luotettavuuden varmistaminen vaatii kuitenkin vielä jatkotutkimuksia.

Avainsanat: Ohjelmistoradio, USRP, RFI, Radioastronomia

AALTO-UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING
PL 13000
00076 AALTO

ABSTRACT OF THE
MASTER'S THESIS

Author: Ville Saari

Title: Monitoring the radio frequency interference in radio astronomical measurements using software radio

Faculty: School of Electrical Engineering, Department of Signal Processing and Acoustics

Date: 19.01.2015 Number of pages: [74 + 37]

Supervisor: Professor Jorma Skyttä

Instructor: M.Sc. Petri Kirves

Language: Finnish

The amount of Radio Frequency Interference (RFI) continues to increase as the number of wireless devices and applications continues to grow. Meanwhile, the spectrum usage continues evolving towards more dynamic operating models, which means that the RFI environment is also evolving. This evolution together with the increasing sensitivity of the receivers used in radio astronomical measurements continues to raise the importance of RFI mitigation and RFI monitoring in radio astronomy.

The objective of this thesis was to implement a control- and data acquisition software for the Ettus Research USRP N210 software defined radio (SDR) for the use of monitoring the RFI in radio astronomical measurements. The goal was to use the SDR to improve the time-domain resolution, sensitivity and flexibility of the RFI monitoring system currently used in the Metsähovi Radio Observatory. Furthermore, the goal was to characterize the technical performance of the USRP N210.

The characterization consisted of measuring the sensitivity, sweep speed, frequency response and bandwidth selectivity of the USRP N210. The performance limiting factors of the system were also examined and documented. Furthermore, the internal error sources and calibration of the USRP were also taken into account while implementing the software.

The software implemented in this thesis can be used to improve the sweep speed and sensitivity of the current RFI monitoring system used in Metsähovi Radio Observatory. However, the reliability of the measurements made with the implemented software needs to be investigated further. Further investigations should also be carried out on the internal error sources and shielding of the USRP N210.

Keywords: Software radio, USRP, RFI, Radio astronomy

Alkusanat

Tämä diplomityö on tehty Aalto-yliopiston sähkötekniikan korkeakoulun signaalinkäsittelyn laitoksella yhteistyössä Metsähovin radiotutkimuslaitoksen kanssa. Työtä ehdotti työn valvojana toiminut professori Jorma Skyttä, jota haluan kiittää työn tarkastamisesta, työn rahallisen tuen mahdollistamisesta sekä työhön liittyvistä arvokkaista neuvoista ja palautteesta. Työn ohjaajana toimi Petri Kirves, jota haluan kiittää työhön ja mittauksiin liittyvistä neuvoista, työn tarkastamisesta sekä kaikesta työn aikanani saamastani tuesta.

Lisäksi haluan kiittää Juha Kallunkia ja Juha Aatrokoskea työssä toteutetun häiriömonitorointijärjestelmän kehitykseen liittyvistä kommenteista sekä järjestelmän asentamiseen saamastani tuesta. Haluan myös kiittää koko Metsähovin henkilökuntaa positiivisesta työympäristöstä sekä kiinnostuksesta diplomityötäni kohtaan. Myös Aalto-yliopiston sähkötekniikan korkeakoulun sekä signaalinkäsittelyn laitoksen henkilökunnat ansaitsevat kiitokset työhön liittyvien yleisten asioiden hoitamisesta.

Suurimmat kiitokseni haluan esittää perheelleni ja ystävilleni kaikesta, koko elämäni aikana saamastani tuesta ja kannustuksesta.

Espoo 19.1.2015

Ville Saari

Sisällysluettelo

Alkusanat ..... **iii**
Sisällysluettelo ..... **iv**
Symbolit ja lyhenteet ..... **vi**
1 Johdanto **1**
1.1 Työn tavoitteet 2
1.2 Työn rakenne 3
2 Radioastronomia ja RFI **4**
2.1 Radioastronomian historiaa 4
2.2 Radioastronomiset mittaukset 6
2.3 Radiotaajuinen allokaatio 6
2.4 Havainnointikeinot 7
2.5 RFI radioastronomiassa 9
2.5.1 Ulkoiset häiriöt 9
2.5.2 Sisäiset häiriöt 10
2.5.3 Häiriöiden vaikutus mittauksiin 11
2.5.4 Häiriöiltä suojautuminen 11
2.6 Häiriöiden mittaamisen perusperiaatteita 12
2.6.1 Spektrianalysaattorien eri tyypit ja toiminta 13
2.6.2 Heterodyne- tyyppisten spektrianalysaattorien mittauskaista 14
2.6.3 Spektrianalysaattorien taajuustason resoluutio ja selektiivisyys 14
2.6.4 Spektrianalysaattorien pyyhkäisyaika ja aikatason resoluutio 16
2.6.5 Herkkyys (DANL-arvo ja kohinaluku) 19
2.6.6 Herkkyyden lisääminen esivahvistimella 20
3 Metsähovin mittausympäristö **21**
3.1 Metsähovin radiotutkimusasema 21
3.2 Metsähovin mittalaitteisto 22
3.3 Häiriömonitorointi Metsähovissa 24
3.3.1 Agilent FieldFox N9912A 26
3.4 Suojautuminen häiriöitä vastaan 26
4 Ohjelmistoradio (SDR) **28**
4.1 Ohjelmistoradion historiaa 28
4.2 Ohjelmistoradion toiminta 29
4.3 Ohjelmistoradion arkkitehtuuri 29
4.3.1 Ohjelmistoradiotyyppien määrittely tasojen mukaan 31
4.4 USRP (Universal Software Radio Peripheral) 32
4.5 Ettus Research USRP N210 33
4.5.1 Emolevy 34
4.5.2 FPGA 34
4.5.3 Kellosignaalit 35
4.5.4 Digitaaliset ylös- ja alasmuuntimet (DUC & DDC) 35
4.5.5 Näytteenotto ja syntetisointi (ADC & DAC) 36
4.5.6 Tytärkortit 36

4.2.7 MIMO 38
5 Häiriömonitoroinnin mittausohjelmisto 39
5.1 GNU Radio 39
5.1.1 GNU Radio -ohjelmien rakenne 39
5.1.2 GNU Radio Companion (GRC) 41
5.1.3 Python–ohjelmointikieli ja NumPy-kirjasto 44
5.1.4 SWIG 44
5.1.5 Python-ohjelmointi GNU Radio:n avulla 44
5.1.6 GNU Radio:n asentaminen 46
5.1.7 GNU Radio versiomuutokset 47
5.2 FPGA:n ohjelmointi 47
5.3 Toteutettu häiriömonitorointiohjelmisto 49
5.4 Järjestelmän rajoitukset, häiriöt ja kalibrointi 51
5.4.1 Järjestelmän taajuusvasteen kalibrointi 51
5.4.2 USRP N210 SBX-tytärkortin kaistanvalintasuodatin 53
5.4.3 USRP N210:n näytteistyksen I/Q epätasapaino 54
5.4.4 USRP N210:n häiriösignaalit 55
5.4.5 USRP N210:n LNA:n epälineaarisuus 58
6 Mittaukset 59
6.1 Järjestelmien aikatason resoluutio 59
6.2 Järjestelmien herkkyys 61
6.3 Järjestelmien kaistanleveyden selektiivisyys 62
6.4 Järjestelmällä toteutettavissa olevat toimintatilat 62
7 Yhteenveto ja tulevaisuuden kehityskohteet 64
7.1 Yhteenveto 64
7.2 Järjestelmän edut ja jatkokehityskohteet 65
Viitteet 68

Symbolit ja lyhenteet

Symbolit

B_{span}	Pyyhkäistävä taajuuskaista [Hz]
dB	Desibeli
dBm	Desibelimilliwatti
dBV	Desibelivoltti
f	Taajuus [Hz]
Δf	Taajuustason resoluutio / kaistanleveys [Hz]
f_{IF}	IF-kaistanpäästösuodattimen keskitaajuus [Hz]
f_{LO}	paikallisoskillaattorin taajuus [Hz]
F_S	Näytteenottotaajuus [Hz]
FFT_{len}	Fourier'n muunnoksen pituus
IF	Välitaajuus [Hz]
G	Tehovahvistus
k	Boltzmannin vakio, 1,38066 x 10^{-23}J/K
N_{FFT}	Viritysaskeleiden lukumäärä
RBW	Resoluutiokaistanleveys [Hz]
T	Lämpötila [K]
t_{AQT}	Näytteenottoon tarvittava aika [s]
t_{PROS}	Yksittäisen FFT:n prosessointiin tarvittava aika [s]

Lyhenteet

ADC	Analog-to-Digital Converter, Analogia-digitaali-muunnin
CR	Cognitive Radio, Kognitiivinen radio
CRAF	Committee on Radio Astronomy Frequencies, Radioastronomiassa hyödynnettävien taajuuksien käyttöä koordinoiva komitea
DC	Direct Current, Tasavirta
DCR	Direct Conversion Radio, Suoramuunnosvastaanotin
DDC	Digital Down Converter, Digitaalinen alasmuunnin
DSP	Digital Signal Processor, Digitaalinen signaaliprosessori

DUC	Digital Up Converter, Digitaalinen ylösmuunnin
ESF	European Science Foundation, Euroopan tiedesäätiö
FFT	Fast Fourier Transform, Nopea Fourier'n –muunnos
FICORA	Finnish Communications Regulatory Authority, Viestintävirasto
FIFO	First In First Out, Dataprotokolla, jossa data sarjamuotoisena lukujonona
FPGA	Field Programmable Gate Array, Ohjelmoitava digitaalinen logiikkapiiri
GPP	General Purpose Prosessor, Yleiskäyttöinen prosessori
GPS	Global Positioning System, Globaali paikannusjärjestelmä
GSM	Global System for Mobile Communications, Toisen sukupolven matkapuhelinjärjestelmä
IC	Integrated Circuit, Integroitu piiri
IF	Intermediate Frequency, Välitaajuus
ITU	International Telecommunication Union, Kansainvälinen televiestintäliitto
LNA	Low Noise Amplifier, Vähäkohinainen vahvistin
LO	Local Oscillator, Paikallisoskillaattori
MAC	Multiply And Accumulate, Digitaalipiirien sisältämä prosessointiyksikkö, joka kykenee samanaikaiseen summa- ja kertolaskun suorittamiseen
MIMO	Multiple-Input and Multiple-Output, Ohjelmistoradiojärjestelmissä käytetty kommunikaatioprotokolla, joka mahdollistaa useiden laitteiden keskinäisen synkronisoinnin
NF	Noise Figure, Kohinaluku
PLL	Phase-Locked Loop, Vaihelukittu silmukka
PSD	Power Spectral Density, Spektrin tehotiheys
Q	Quadrature-phase, 90-asteen vaihesiirrossa oleva signaali
RBW	Resolution BandWidth, Resoluutiokaistanleveys
RF	Radio Frequency, Radiotaajuus
RFI	Radio Frequency Interference, Radiotaajuinen häiriö
SCPI	Standard Commands for Programmable Instruments, Ohjelmoitavien instrumenttien standardikomennot
SDR	Software-Defined Radio, Ohjelmistoradio
SNR	Signal-to-Noise Ratio, Signaali-kohinasuhde
SPI	Serial Peripheral Interface, Sarjamuotoinen liitäntärajapinta
SR	Software Radio, Ohjelmistoradio
TX	Transmitter, Lähetin

USB	Universal Serial Bus, Yleiskäyttöinen sarjamuotoinen liitäntä
USRP	Universal Software Radio Peripheral, Yleiskäyttöinen ohjelmistoradiolaite
VCO	Voltage Controlled Oscillator, Jänniteohjattu oskillaattori
VGA	Variable Gain Amplifier, Ohjattava vahvistin
VHDL	Very high speed Integrated circuits Hardware Description Language, FPGA-piirien ohjelmointiin käytetty laitteistokuvauskieli
VLBI	Very Long Baseline Interferometry, Pitkäkantainterferometria
WiFi	Wireless Fidelity, Langaton lähiverkkoteknologia
WLAN	Wireless Local Area Network, Langaton lähiverkko

Luku 1

1 Johdanto

Radioastronomia on tähtitieteen tutkimuksen osa-alue, joka keskittyy maailmankaikkeuden fyysisen olemuksen tutkimiseen avaruudesta saapuvan sähkömagneettisen säteilyn avulla. Radioastronomiassa mitattavat signaalit ovat mittauslaitteiston ja mitattavien ilmiöiden suuresta välimatkasta aiheutuvan avaruudellisen vaimennuksen ansiosta hyvin heikkoja, mistä johtuen niiden vastaanottamiseen tarvitaankin erittäin herkkiä radiovastaanottimia. Lisäksi vastaanotettuja signaaleja joudutaan vahvistamaan, jotta niiden voimakkuus saadaan sopivalle tasolle niiden näytteistystä varten. Mitattavan säteilyn heikko tehotaso tekee radioastronomisista mittauksista hyvin herkkiä häiriösignaaleille, joita voi kytkeytyä mittauskokoonpanoon mittausten ulkopuolisista häiriölähteistä. [1]

Maan pinnalta suoritettavissa radioastronomisissa mittauksissa normaalioloissa käytettävissä olevaa noin 30 MHz – 300 GHz taajuusikkunaa rajoittaa ilmakehän vaimennus, joka on riippuvainen Auringon aktiivisuudesta sekä avaruudesta saapuvan säteilyn tulokulmasta [2]. Radioastronomissa mittauksissa käytettävät vastaanottimet toimivat yleensä radioastronomisiin mittauksiin varatuilla taajuuskaistoilla korkeilla kymmenien tai satojen gigahertsien taajuuksilla. Kyseisillä taajuuskaistoilla on kuitenkin entistä enemmän myös muita käyttäjiä, eivätkä kaistoille määrätyt käyttökiellot välttämättä takaa radioastronomisten mittausten häiriöttömyyttä [3, pp. 100-102]. Lisäksi A/D-muuntimien rajallinen näytteistysnopeus rajoittaa suoraan näytteistettävissä olevien signaalien taajuusaluetta. Signaalien näytteistys joudutaankin useimmiten suorittamaan vastaanottimien alasmuutetulla IF-välitaajuudella, mikä laajentaa mittauksia mahdollisesti häiritsevien signaalien taajuuskaistaa. [4] [5]

Radioastronomiset mittaukset suoritetaan yleensä radiohiljaisilla alueilla, joissa radiotaajuisilla yhteyksillä toimivien laitteiden käyttö on kielletty. Häiriösignaaleja kytkeytyy kuitenkin mittauksiin alueen ulkopuolelta, yksityishenkilöiden käytössä olevista elektronisista laitteista, kuten matkapuhelimista, sekä viranomaispuolelta, esimerkiksi tutka- ja lentoliikenteestä [6]. Lisäksi mittauksia häiritseviä häiriösignaaleja syntyy alueen sisällä käytössä olevien elektronisten laitteiden, kuten mittalaitteiden, mikroaaltouunien, tietokoneiden, hakkurivirtalähteiden sekä sähkömoottorien toiminnan yhteydessä. [1] [7]

Häiriöiltä suojautuminen ja häiriöiden suodattaminen ovatkin entistä tärkeämpiä tutkimuskohteita radioastronomian alalla, sillä mittauksissa käytetyt kehittyneet vastaanottimet ovat aiempaa herkempiä myös häiriöitä kohtaan [8]. Häiriöt voivat lisäksi vaikuttaa mittauksiin, vaikka ne sijaitsisivat selvästi itse mittauskaistan ulkopuolella [9]. Mittauksia ympäröivän häiriöympäristön tunteminen onkin tärkeää, jotta mittausten luotettavuus voidaan säilyttää. Lisäksi useimmat häiriöiden suodattamiseen käytetyt menetelmät pohjautuvat häiriöiden tunnistamiseen, mikä korostaa häiriömonitoroinnin tärkeyttä myös häiriöiden suodattamisen osalta. [8]

Langattoman kommunikaation alan voimakas kasvu lisää ja monimutkaistaa radioastronomisiin mittauksiin kytkeytyviä häiriöitä. Langattomia yhteyksiä hyödyntäviä laitteita,

kuten matkapuhelimia, kannettavia tietokoneita sekä näiden oheislaitteita käytetään entistä enemmän kuluttajaelektroniikassa, teollisuudessa sekä erilaisissa palveluissa, kuten automatisoiduissa tehtaissa ja etähoitotyökaluissa. Lisäksi laitteiden käyttämät nopeat yhteydet vaativat käyttöönsä entistä leveämpiä taajuuskaistoja [10]. Laitteiden lisääntyminen asettaakin suuria haasteita käytettävän spektrin jakamiselle. [5] [11] [12]

Nykyinen spektrin regulaatio perustuu staattiseen taajuusallokaatiopolitiikkaan. Radiotaajuinen spektri on siis jaettu taajuusalueisiin, joiden käyttöoikeudet on määrätty tietylle käyttökohteelle tai järjestelmälle. Staattinen allokaatio tuottaa varsin epätasaisen spektrin käyttöasteen, joka vaihtelee selvästi taajuusalueen, ajan sekä paikan funktiona. Langattomia yhteyksiä hyödyntävien laitteiden lukumäärän lisääntyminen onkin johtanut tiettyjen taajuusalueiden vapaiden kaistojen täyttymiseen ja päällekkäiseen kaistankäyttöön, jossa yksittäinen taajuuskaista on useampien laitteiden käytössä samanaikaisesti. [5] [11] [12]

Eräänä ratkaisuna päällekkäisen kaistankäytön välttämiseen on esitetty nykyisten säännösten uudistamista kognitiivisiin radiojärjestelmiin perustuvan dynaamisen spektrinkäytön avulla. Kognitiivinen radio (engl. Cognitive Radio, CR) on Joseph Mitolan kehittämä termi, joka viittaa ympäristöstään tietoiseen, älykkääseen oppivaan radiojärjestelmään. Dynaamisella spektrinkäytöllä taas tarkoitetaan tilannetta, jossa järjestelmät mittaavat ja valitsevat itse käyttämänsä taajuusalueen tehostaen näin spektrin käyttöastetta. [5] [11] [12] [13]

Kognitiiviseen radioon perustuvat dynaamisesti ja itsenäisesti toimintataajuuttaan muuttavat järjestelmät tulevatkin todennäköisesti yleistymään tulevaisuudessa. Radioastronomisten mittausten näkökulmasta tämä tarkoittaa sitä, että häiriösignaalien lisääntymisen ohessa myös niiden käyttämien taajuusalueiden ennustaminen muuttuu entistä haastavammaksi. Tämä asettaa vaatimuksia myös radiohäiriöiden monitorointiin käytettävälle järjestelmälle, sillä sen täytyy pystyä havaitsemaan entistä nopeammin muuttuvia häiriösignaaleja [12].

Tässä työssä toteutetulla ohjelmistoradioon perustuvalla häiriömonitorointijärjestelmällä pyritään vastaamaan edellä kuvattuihin haasteisiin. Työn dokumentoinnilla pyritään antamaan monipuolinen yleiskuva häiriömonitoroinnista, ohjelmistoradiojärjestelmistä, työssä käytetystä Ettus Research N210 –ohjelmistoradiolaitteesta sekä sille toteutetuista ohjelmistoista. Lisäksi dokumentoinnilla pyritään yksinkertaistamaan järjestelmän jatkokehittämistä.

1.1 Työn tavoitteet

Tämän diplomityön tarkoitus on toteuttaa Metsähovin radio-observatoriolle ohjelmisto aiemmin hankitulle ohjelmistoradiojärjestelmälle radioastronomisten mittausten häiriötekijöiden monitorointiin. Toteutetun ohjelmiston avulla pyritään kehittämään ja tukemaan Metsähovissa nykyisin käytössä olevan RFI-monitorointijärjestelmän toimintaa. Ohjelmistoradiojärjestelmän käytöllä pyritään mahdollistamaan entistä tarkempi nopeiden signaalien havaitseminen. Työn tarkoituksena on myös pohtia järjestelmän tarjoamia mahdollisuuksia häiriömonitoroinnin jatkokehityksen sekä järjestelmän toiminnan integroitavuuden osalta.

Tämä työ on jatkumoa Jukka-Pekka Porkon vuonna 2010 Metsähoviin tekemälle diplomityölle ”Radio frequency interference in radio astronomy” [1]. Kyseinen työ koostui pääosin Metsähovin nykyisen häiriöympäristön tutkimisesta, keskittyen radioastronomisissa mittauksissa käytettävillä IF-välitaajuuksilla esiintyviin häiriöihin. Työssä mitattiin Metsähovin sisäistä ja ulkoista häiriöympäristöä 30 - 2600 MHz:n taajuusalueella kiinnittäen erityistä huomiota observatorion sisäisesti tuotettuihin häiriöihin. Työssä suoritetuissa mittauksissa löydettiin useita häiriölähteitä ja työssä esitettiin useita tekniikoita ja menetelmiä kyseisiltä häiriöiltä suojautumiseen. Työssä korostettiin radio-observatorion ulkoisen ja sisäisen häiriöympäristön monitoroinnin tärkeyttä radioastronomisten mittausten tulosten paikkansapitävyyden ja luotettavuuden säilyttämisessä.

1.2 Työn rakenne

Diplomityön luvussa kaksi esitetään lyhyt katsaus radioastronomian historiaan, radioastronomisiin mittauksiin ja näitä häiritseviin radiotaajuisiin häiriösignaaleihin sekä kyseisten signaalien suodattamiseen. Lisäksi luvussa perehdytään itse häiriöiden mittaamiseen keskittymällä lähinnä pyyhkäisevien-, FFT- sekä hybridi- tyyppisten spektrianalysaattorien toiminnan esittelyyn. Tällä pyritään selventämään Metsähovissa nykyisin häiriömonitoroinnissa käytettävän spektrianalysaattorin, sekä tässä työssä toteutetun ohjelmistoradiojärjestelmän toimintaa. Luvussa kolme esitellään Metsähovin observatorioalueen tutkimustoiminnan, radioastronomian tutkimuksessa käytettävän laitteiston sekä Metsähovin mittausympäristön pääpiirteet. Luvussa neljä esitetään ohjelmistoradion konsepti lähtien liikkeelle itse ohjelmistoradion termin historiasta, ohjelmistojärjestelmien luokittelusta sekä kyseisten luokkien keskinäisistä eroista. Luvussa myös vertaillaan muutamia nykypäiväisiä ohjelmistoradiojärjestelmiä niiden komponenttien, suorituskyvyn sekä hinnan suhteen. Lisäksi luvussa esitellään työssä käytettävän Ettus Research N210 ohjelmistoradion laitteistopuolta ja sen yksittäisten osakokonaisuuksien toiminnan perusperiaatteita. Luku viisi sisältää työn ohjelmistototeutuksen esittelyn. Lisäksi luvussa esitetään yleisesti ohjeita ohjelmistoradion ohjelmistokehitykseen tarvittavien työkalujen asentamisesta lähtien. Lopuksi, luvussa kuusi esitetään työssä suoritetut mittaukset ja saadut mittaustulokset, jonka jälkeen luvussa seitsemän esitetään diplomityön yhteenveto.

Luku 2

2 Radioastronomia ja RFI

Tässä luvussa käydään läpi radioastronomian perusperiaatteita lähtien liikkeelle radioastronomisen tutkimuksen historiasta, nykyään suoritettavien radioastronomisten mittausten pääpiirteistä sekä mittauksia rajoittavista tekijöistä. Tämän jälkeen luvussa käydään läpi radiotaajuisten häiriösignaalien eri tyypit ja niiden vaikutus itse radioastronomisiin mittauksiin. Lisäksi luvun lopussa esitetään häiriösignaalien mittaamisen perusperiaatteita.

2.1 Radioastronomian historiaa

Radioastronomia, eli radiotähtitiede on suhteellisen uusi tieteenala, sillä se sai alkunsa vasta 1930-luvulla, kun Bell Telephone Labs:lla radioinsinöörinä työskennellyt Karl Jansky määrättiin selvittämään yhtiön Atlantin ylittävän radiopuhelinyhteyden toimintaa häiritsevien signaalien alkuperää. Jansky suoritti mittaukset rakentamansa käännettävän 20,5 MHz:n taajuudella toimivan antennisysteemin avulla ja esitteli mittaustensa tulokset kansainvälisen radiotieteen unionin kokouksessa vuonna 1932. Hän luokitteli mittauksissa havaitsemansa häiriösignaalit paikallisiin- ja kaukaisiin ukkoshäiriöhin sekä tasaiseen, tuntemattomasta lähteestä peräisin olevaan kohinaan. Jansky arveli kohinan olevan peräisin Auringosta, sillä sen intensiteetillä oli selvä korrelaatio Maan pyörimisliikkeen kanssa. Hänen myöhemmät mittauksensa kuitenkin osoittivat, että kohina oli peräisin Linnunradan keskustan suunnasta [14]. Janskyn löydöt eivät kuitenkaan saaneet erityistä huomiota tieteellisessä yhteisössä, eikä tutkimuksia Linnunradan säteilyyn liittyen jatkettu hänen pyynnöistään huolimatta, sillä alkuperäisen tutkimuksen päämäärä katsottiin saavutetuksi [15]. [1] [2] [16]

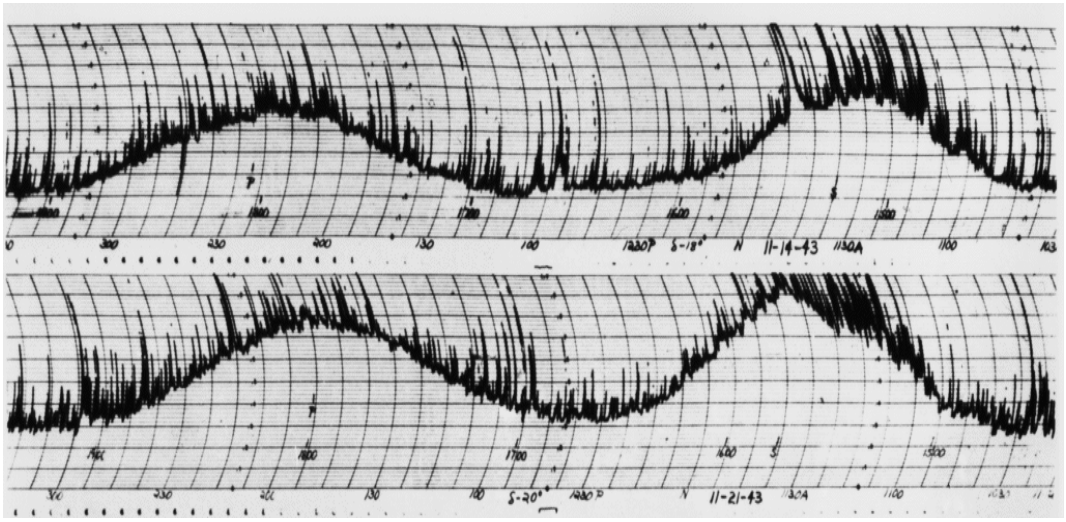
Toinen radioastronomian pioneeri oli Grote Reber. Hän kiinnostui Janskyn löytämästä Linnunradan säteilystä ja halusi selvittää tarkemmin, mistä havaitut signaalit olivat peräisin ja mikä ilmiö kyseiset signaalit aiheutti. Reber jatkoikin Janskyn työtä rakentamalla mittauksiaan varten 9,5 metrin parabolisen radioteleskoopin takapihalleen Wheatoniin Illinnoisiin. Reberin ensimmäiset 3,3 GHz:n ja 900 MHz:n taajuuksilla suorittamat mittaukset epäonnistuivat lähinnä laitteistoteknisistä syistä. Hän onnistuikin vahvistamaan Janskyn tulokset vasta vuonna 1938 käyttäen kolmatta rakentamaansa radiovastaanotintaan ja 160 MHz:n taajuusaluetta. Reber jatkoi mittauksiaan vuosia ja valmisti niiden pohjalta ensimmäisen koko taivaan kattavan radiokartan, joka julkaistiin vuonna 1944. Astronominen tiedeyhteisö oli aluksi skeptinen Reberin löytöjä kohtaan, mutta toisen maailmansodan jälkeen entistä useammat tutkimusryhmät ympäri maapalloa alkoivat rakentaa entistä suurempia ja herkempiä antenneja ja vastaanottimia Janskyn ja Reberin pohjatyön jatkamiseksi. Reberin havaintoja pidetäänkin usein koko havaitsevan radioastronomian alkuna. [15] [16] [17]

Vuonna 1944 Henrik van de Hulst ennusti vedyn emissiolinjan olemassaolon. Hänen ennustuksensa mukaan spektrissä tulisi esiintyä 21,1 cm:n aallonpituudella vetyatomin elektronin spinin kääntymisestä johtuva spektriviiva. Harold Ewen ja Edward Purcell

havaitsivatkin kyseisen emissioviivan myöhemmin vuonna 1951. Emissioviivan avulla oli mahdollista kartoittaa vedyn jakaumaa Linnunradalla, minkä pohjalta oli mahdollista tehdä arvioita itse Linnunradan rakenteesta. Mitatun säteilyn yhdistäminen kohteen ainerakenteen- ja dynaamisten ominaisuuksien tutkimiseen syvensivät mittausten merkitystä huomattavasti. Tämä osiltaan johti radioastronomian kiinnostuksen nopeaan kasvuun 1960-luvulla. Kyseisen ajanjakson suurimpiin löytöihin kuuluvat Maarten Schmidtin vuonna 1963 löytämät kvasaarit, Arno Penziaksen ja Robert Wilsonin vuonna 1965 löytämä kolmen kelvinin taustasäteily, sekä Jocelyn Bellin ja Antony Hewishin vuonna 1967 löytämät pulsarit. [16] [18]

Radioastronomisissa mittauksissa käytettävien laitteiden tekninen suorituskyky on ollut historiallisesti eräs suurimmista radioastronomian kehitystä rajoittaneista tekijöistä. Suurimmat radioastronomiset löydöt ovatkin useimmiten liittyneet uusien tekniikoiden tai laitteiden kehitykseen. Moderni teknologia on mahdollistanut erittäin herkkien radiovastaanottimien rakentamisen ja nykyisin radioastronomista säteilyä voidaankin mitata koko havaittavan maailmankaikkeuden reunoilta asti. Mittauksissa käytettyjen radioteleskooppien fyysinen koko on suurentunut huomattavasti entistä parempaa erottelukykyä ja herkkyyttä tavoiteltaessa. Suurimpien kiinteiden teleskooppien vastaanotinpeilien halkaisijat ovatkin nykyään jo satojen metrien kokoluokassa. Interferometrisiä, useampiin vastaanotinantenneihin perustuvia tekniikoita käyttäen on kuitenkin mahdollista syntetisoida reilusti edellä mainittuja suurempiakin radioteleskooppirakenteita, mikä on avannut uusia kehityskohteita ja osittain mahdollistanut vastaanotinlaitteiden fyysisten rajoitusten ohittamisen. [1] [19]

Myös radiotaajuiset häiriöt ovat rajoittaneet radioastronomian kehitystä ja jopa Grote Reber joutui jo 1930-luvulla tekemissään mittauksissa kohtaamaan radiotaajuisia häiriösignaaleja nykypäivän radioastronomien tapaan. Hän asui Chicagon esikaupunkialueella, jossa autoliikenteestä peräisin olevat häiriöt rajoittivat hänen työskentelynsä yöaikaan [1]. Kuvassa 2.1 nähdään kyseisten häiriösignaalien vaikutus Reberin vuonna 1943 suorittamissa mittauksissa. Signaalin laakeat vaihtelut ovat peräisin Linnunradalta saapuneesta astronomisesta säteilystä, kun taas kuvaajan suuriamplitudiset nopeat piikit ovat peräisin autojen sytytystulppien kipinöistä [17].



Kuva 2.1: Häiriöitä Grobe Reber:n suorittamissa radioastronomisissa mittauksissa. Julkaistu vuonna 1944 "Astrophysical Journal" lehden numero 100, sivulla 278. [17]

2.2 Radioastronomiset mittaukset

Radioastronomisissa mittauksissa käytetään erittäin herkkiä radiovastaanottimia kaukaisten taivaallisten kohteiden, kuten kvasaarien ja pulsarien, sekä läheisempien kohteiden, kuten Auringon aikaansaamien radioemissioiden havainnointiin ja tutkimiseen. Maapallon ilmakehän aiheuttama vaimennus rajoittaa maasta käsin tehtävissä radioastronomissa mittauksissa vastaanotettavien signaalien taajuuskaistaa. Ilmakehän ionosfäärin vapaat elektronit heijastavat taajuudeltaan alle 30 MHz:n radioaallot takaisin avaruuteen, kun taas ylärajan vastaanotettavien signaalien taajuudelle asettavat ilmakehän kaasumolekyylit, joiden aiheuttama vaimennus kasvaa yli 300 GHz:n taajuuksilla huomattavasti. Kyseisen taajuusalueen sisällä radioastronomisissa mittauksissa käytettävissä olevaa taajuuskaistaa rajoittaa lähinnä mittausten ulkopuolinen häiriöympäristö. [1] [2]

Radioastronomiset mittaukset suoritetaankin yleensä pelkästään niille varatuilla suojatuilla taajuuskaistoilla. Kyseisille taajuuskaistoille voi kuitenkin kytkeytyä häiriöitä esimerkiksi vioittuneista laitteista. Radioastronomiset mittaukset suoritetaan yleensä myös varsin korkeilla taajuuksilla ja mitatut signaalit joudutaan sekoittamaan alemmille IF-välitaajuuksille niiden näytteistämisen mahdollistamiseksi. Tämä laajentaa mittauksia mahdollisesti häiritsevien signaalien taajuuskaistaa. Lisäksi jatkuvasti kasvava radiokäyttökohteiden sekä radiotaajuuksilla toimivien laitteiden määrä vaikeuttaa radioastronomisissa mittauksissa käytettävien taajuusalueiden sisältämien häiriösignaalien suodattamista. [1] [2]

2.3 Radiotaajuinen allokaatio

Radioastronomiassa mittauksia häiritsevät signaalit voivat olla tehotasoiltaan jopa yli miljardikertaisia itse mitattaviin signaaleihin verrattuna [8]. Radioastronomisten mittausten suojaamiseksi niiden suorittamiseen onkin varattu taajuuskaistoja, joilla muu radioliikenne on kielletty. Varaukset on tehnyt Kansainvälinen televiestintäliitto (engl. International Telecommunication Union, ITU), joka on koko maailman radiotaajuisen spektrin koordinoinnista vastaava hallintoelin. Suojatut taajuuskaistat ovat radioastronomisten mittausten kannalta välttämättömiä, sillä ne vähentävät oleellisesti mittauksiin kytkeytyvien häiriösignaalien määrää. Samalla ne mahdollistavat samanaikaisiin, eri paikoissa suoritettaviin mittauksiin perustuvien interferometristen mittausten yhtenäisen suorittamisen koko maapallon alalla. [1] [2]

Radioastronomiassa lähes mikä tahansa taajuusalue sisältää mittausten kannalta hyödyllistä tietoa. Suojattujen kaistojen lisäämistä rajoittaa kuitenkin kaupallisten, taloudellisesti tuottavien radiokäyttökohteiden määrän jatkuva kasvu [2]. Sergei Gulyaev ja Paul Banks ovat lisäksi osoittaneet julkaisussaan ”Radio sky and the right to observe it” [20], että nykyiset suojattujen kaistojen määritelmät ovat vajavaisia, sillä ne eivät huomioi riittävästi esimerkiksi kohteiden punasiirtymästä johtuvaa taajuusalueen muutosta. He ovatkin esittäneet, että suojattujen taajuuskaistojen allokaatiota tulisi uudistaa huomioiden kyseiset ongelmakohdat. Radioastronomisiin mittauksiin varattujen kaistojen valitsemisperusteista ja tarpeellisuudesta on mahdollista löytää lisää tietoa esimerkiksi Euroopan tiedesäätiön, eli ESF:n CRAF-toimikunnan julkaisusta ”CRAF Handbook for Radio Astronomy” [3].

Suomessa radiotaajuisen spektrin käyttöä valvoo Viestintävirasto, jonka määrittelemä taajuusallokaatio kattaa 9 kHz:n ja 3000 GHz:n välisen taajuusalueen. Käytettävissä oleva taajuusalue on siis erittäin laaja ja taajuuskaistan voisi kuvitella riittävän hyvin kaikkiin nykyisiin käyttökohteisiin. Sääilmiöt ja ilmakehän vaimennus heikentävät kuitenkin radiosignaalien etenemisominaisuuksia yli kymmenen gigahertsin taajuuksilla, mikä rajoittaa sitä korkeampien taajuusalueiden käyttöä. Korkeilla taajuuksilla toimivat laitteet ovat myös rakenteeltaan monimutkaisia ja kalliita. Spektrin käyttö onkin yleisesti käytössä olevissa kohteissa keskittynyt alle 80 GHz:n taajuuksille, eli koko hallinnoidusta spektristä on käytössä vain noin kolmen prosentin osuus. Lisäksi kaikista Suomessa käytössä olevista radiolaitteista noin 95 prosenttia toimii alle 10 GHz:n taajuuksilla ja jopa 99 prosenttia alle 25 GHz:n taajuuksilla. Radioastronomisten mittausten kannalta tämä tarkoittaa sitä, että suurin osa vähähäiriöisistä taajuusalueista sijaitsee hyvin korkeilla taajuuksilla, kun taas alemmilla taajuuksilla, ja varsinkin näytteistykseen käytettävillä IF-välitaajuuksilla, muita taajuusalueen käyttäjiä ja käyttäjistä aiheutuvia häiriöitä on paljon. [21]

2.4 Havainnointikeinot

Radioastronomiassa yleisimmin käytettävät havainnointi- eli observaatiokeinot ovat kontinuumi-, spektriviiva-, aurinko- ja interferometriset observaatiot. Edellä mainituista kontinuumi-, spektriviiva- ja aurinko-observaatiot voidaan suorittaa yhden radioteleskoopin avulla, kun taas interferometriset mittaukset perustuvat useammilla vastaanotinantenneilla saatujen mittaustulosten yhdistämiseen. Useampia vastaanottoantenneja käyttämällä saavutetaan observaatiotyypistä riippumatta parempi herkkyys ja/tai resoluutio yhteen antenniin perustuviin järjestelmiin verrattuna. Interferometrisiin mittauksiin tarvittavat järjestelmät ovat kuitenkin kokoonpanoltaan monimutkaisia ja kalliita. [1] [2]

Kaikki radioastronomiset havainnointikeinot perustuvat radiotaajuisen sähkömagneettisen säteilyn eri taajuuksilla olevien osien energiavuon tiheyden eli sähkömagneettisen spektrin tutkimiseen. Kontinuumi- eli monitaajuusspektristä voidaan nähdä lähteen säteilyn eri taajuisten osien energia samanaikaisesti. Monitaajuusspektriä tutkimalla onkin mahdollista saada tietoa säteilyn syntymekanismeista. Spektriviiva-observaatiot taas perustuvat aineen spektrin absorptio- ja emissioviivojen tutkimiseen. Eri aineet absorboivat, eli vaimentavat, sekä emissoivat, eli säteilevät, sähkömagneettista säteilyä eri tavoin, minkä ansiosta kohteen spektriviivoja tutkimalla voidaan saada tietoa sen aine- ja molekyylirakenteesta. Lisäksi spektriviivan Doppler-siirtymää tutkimalla on mahdollista laskea kohteen sisältämän materian liikesuunta, nopeus, lämpötila sekä säteilyteho. [1] [22]

Aurinko on voimakkain Maasta käsin havaittava säteilylähde, jonka tuottaman säteilyn taajuusalue ulottuu radiotaajuuksilta aina röntgensäteilyyn asti. Eri taajuuksilla voidaan mitata Auringossa eri syvyyksillä tapahtuvia ilmiöitä ja itse mittauksissa käytettävissä olevaa taajuuskaistaa rajoittaa vain mittausten ulkopuolinen häiriöliikenne. Radioastronomiset Aurinko-observaatiot koostuvat suurilta osin Auringon aktiivisuuden seuraamiseen ja kartoitukseen liittyvistä mittauksista. Aurinko onkin erittäin tärkeä havaintokohde, sillä sitä tutkimalla on mahdollista saada tarkkaa tietoa yleisesti tähtien pyörimiseen, aktiivisuusilmiöihin, magneettikenttiin sekä pinnan rakenteisiin liittyen. [16]

Kaikki radioastronomiassa mitattavat kohteet säteilevät useiden eri säteilymekanismien tuottamia radiosäteilytyyppejä. Lisäksi mittauksissa on aina näkyvissä taustasäteilystä johtuva komponentti. Säteilymekanismien monipuolisuudesta johtuen kontinuumi- ja spektriviiva-obsevaatioissa mitattavat kohteet kattavatkin kaiken aina omasta Aurinkokunnastamme kaukaisimpiin kvasaareihin ja koko havaittavan universumin reunoille asti. Mitattavasta kohteesta vastaanotettava radiotaajuinen spektri riippuu kuitenkin aina kohteen fyysisistä ominaisuuksista, kuten lämpötilasta, liiketilasta ja ainerakenteesta. Kohteen tilan muutokset vaikuttavat sen radiosäteilyn ominaisuuksiin, kuten spektriin ja polarisaatioon, mistä johtuen näissä ominaisuuksissa voidaankin usein havaita varsin merkittäviä ajallisia vaihteluja. Taulukossa 2-1 on esitetty eri säteilylajien jaottelu ja kyseistä säteilylajia säteilevät tyypilliset radioastronomiassa mitattavat kohteet. [2] [23, pp. 19-24]

	Säteilyn aiheuttaja	**Säteilyn tyyppi**	**Säteilylähteet**
Lämpösäteily	Aineen molekyylien lämpöliike	kontinuumi-säteily	Planeetat ja kuut sekä kosminen 3 kelvinin taustasäteily
Syklotroni-säteily	Magneetti- tai sähkökentän kiihdyttämä varattu hiukkanen	kontinuumi-säteily	Tähtienväliset molekyylipilvet ja pöly, mustat aukot
Synkrotroni-säteily	Magneettikentän kiihdyttämän relativistisen elektronin liikesuuntaansa aiheuttamaa säteilyä.	kontinuumi-säteily	Planeetat ja kuut, tähtienväliset molekyylipilvet ja pöly, supernovajäänteet, pulsarit, radiogalaksit sekä kvasaarit
Terminen jarrutussäteily	Varautuneen hiukkasen nopeuden muutokset	kontinuumi-säteily	Kuumat ionisoituneen vedyn alueet
Masersäteily	Atomien pitkäaikaisen viritystilan purkautuminen. Emissiot mm. 1, 6, 8 ja 22 GHz:llä	spektriviiva-säteily	Tähtienväliset molekyylipilvet ja pöly, pulsarit
Atomien ja molekyylien absorption ja emission spektriviivat	Atomin energiatilan tai molekyylin rotaation muutos. (ilman Doppler-siirtymiä kapeakaistaisia, emissiospektriviivoja)	spektriviiva-säteily	Tähtienväliset molekyylipilvet ja pöly, komeetat

Taulukko 2-1: Radioastronomiassa mitattavat säteilylajit, muokattu lähteestä [2]

2.5 RFI radioastronomiassa

Radioastronomia perustuu passiiviseen mittaukseen, jossa mitattavan, ulkoavaruudesta saapuvan säteilyn tehotaso voi tietyissä tapauksissa olla vain miljardisosan luokkaa verrattuna mittauksia häiritsevien signaalien tehotasoon. Lisäksi radioastronomiassa käytettävien vastaanottimien resoluutio ja herkkyys kehittyvät jatkuvasti, mistä johtuen yhä pienemmät häiriöt aiheuttavat virheitä itse mittauksiin. Tämä tekee virheettömien mittausten suorittamisesta entistä monimutkaisempaa. Häiriösignaalien havaitsemiseen ja suodattamiseen keskittyvät menetelmät ovatkin nykyään eräs keskeisimmistä tutkimuskohteista radioastronomian tieteenalalla. [8]

Radioastronomissa mittauksissa esiintyvät häiriösignaalit voidaan jakaa ulkoisiin ja sisäisiin häiriösignaaleihin. Ulkoisilla häiriösignaaleilla tarkoitetaan radio-observatorion ulkopuolelta mittauksiin kytkeytyviä jatkuvia häiriösignaaleja, kuten televisio- ja radiolähetyksiä, sekä nopeasti muuttuvia häiriösignaaleja, kuten viestintä-, tutka- ja lentoliikenteen aiheuttamia signaaleja. Sisäisillä häiriösignaaleilla taas tarkoitetaan radio-observatoriossa käytettävien elektronisten laitteiden, kuten mikroaaltouunien, tietokoneiden sekä digitaalisten mittalaitteiden aiheuttamia signaaleja. [1] [24]

Seuraavissa luvun osissa käydään läpi kyseisten häiriötyyppien ominaisuuksia ja niiden vaikutuksia itse mittauksiin. Lukijaa suositellaan tutustumaan luvussa 2.5 esitettyihin aiheisiin vielä tarkemmin esimerkiksi Jukka-Pekka Porkon ”Radio frequency interference in radio astronomy”-diplomityön kautta. Kyseinen työ on löydettävissä lähteestä [1].

2.5.1 Ulkoiset häiriöt

Ulkoisia häiriöitä aiheuttavat lähteet voidaan jakaa niiden maantieteellisen sijainnin mukaisesti paikallisiin, alueellisiin ja globaaleihin häiriölähteisiin [24]. Häiriöt voidaan myös jakaa itse häiriölähteen tyyppien mukaan maaperäisiin sekä ilma- ja avaruusperäisiin lähteisiin [1]. Globaalien, ilma- ja avaruusperäisten häiriölähteiden, kuten satelliittikommunikaatiopalvelujen ja ilmailuliikenteen aiheuttamien häiriöiden torjuminen on hankalaa, sillä itse häiriölähteisiin on yleensä lähes mahdotonta vaikuttaa [24]. Tämän tyyppisten häiriölähteiden aiheuttamien signaalien ja mittauksissa käytettävien instrumenttien välisestä suuresta etäisyydestä aiheutuva avaruudellinen vaimentuminen heikentää kuitenkin häiriöiden tehotasoa huomattavasti. Tämä pienentää häiriöiden kytkeytymisen mahdollisuutta varsinkin vastaanottimen alasmuutetun IF-välikaistan osalta. Ilma- ja avaruusperäiset signaalit voivat tosin kytkeytymissuuntansa ansiosta kytkeytyä mittauksiin myös vastaanotinantennin pääkeilan kautta, jolloin ne voivat ylikuormittaa vastaanotinta. Ilma- ja avaruusperäiset häiriöt ovatkin erityisen haitallisia aktiivisten lentoliikenteen reittien lähellä sijaitseville radio-observatorioille. [1] [24] [25]

Paikalliset ja alueelliset, maaperäiset häiriölähteet sijaitsevat suhteellisen lähellä itse radio-observatoriota, joten niiden aiheuttamien häiriösignaalien tehotaso on selvästi suurempi verrattuna ilma- ja avaruusperäisiin häiriöihin. Jatkuvat alueelliset häiriösignaalit, kuten TV-lähetyksistä aiheutuvat signaalit, ovat kuitenkin tehotasoiltaan ja taajuuskaistaltaan varsin hyvin tunnettuja, mikä yksinkertaistaa niiltä suojautumista. Tämän tyyppisten häiriölähteiden vaikutusta mittauksiin voidaankin vähentää huomattavasti suorittamalla mittaukset radioastronomian käyttöön varatuilla taajuusalueilla. Nopeasti muuttuviin häiriöihin varautuminen on sen sijaan varsin monimutkaista.

Nopeasti muuttuvia häiriöitä voidaankin suodattaa lähinnä pidentämällä mittauksissa käytettyjä integrointiaikoja sekä suodattamalla mitattua signaalia esimerkiksi adaptiivisten suodatusmenetelmien avulla. [1] [24]

2.5.2 Sisäiset häiriöt

Radioastronomissa observatorioissa käytettävät elektroniset laitteet, kuten tietokoneet, tieteelliset instrumentit, mikroaaltouunit ja sähkömoottorit voivat tuottaa radiotaajuisia häiriösignaaleja niiden sisältämien antenneina toimivien rakenteiden, korkeiden jännitteiden sekä heikon sisäisen suojaustason ansiosta. Langattomat yhteydet, kuten matkapuhelinyhteydet, Bluetooth tai WLAN voivat aiheuttaa mittausten IF-välikaistalle voimakkaita, häiriöteholtaan jopa 1 - 100 mW:n tasoisia häiriöitä. Loisteputkivalot ja heikkolaatuiset LED-valojen virtalähteet voivat aiheuttaa laajakaistaisia häiriöitä, minkä ansiosta niiden käyttöä tulisi välttää. Ulkoisten häiriöiden tapaan sisäiset häiriösignaalit voivat olla jatkuvia tai nopeasti muuttuvia. Radio-observatorioiden sisäisten häiriöiden vaikutus radioastronomisiin mittauksiin voi olla huomattava, sillä signaalit syntyvät lähellä mittauksiin käytettävää laitteistoa, joten niiden etäisyydestä riippuvat häviöt ovat pieniä. [1] [26]

Radio-observatorioiden sisäisiä häiriöitä aiheuttavat laitteet ovat usein tarpeellisia itse mittausten tekemiseen, joten niiden suora käytöstä poistaminen ei useimmiten ole mahdollista. Mittauksiin kytkeytyvien häiriösignaalien määrää voidaan kuitenkin vähentää huomattavasti parantamalla laitteiden ja mittauksissa käytettävien instrumenttien välistä häiriösuojaustasoa. Taloudellisista ja käytännöllisistä syistä johtuen tämä ei kuitenkaan aina ole mahdollista. Häiriöitä tuottavien laitteiden tunnistaminen onkin tärkeää mittausten häiriösuojauksen kehittämisen kustannustehokkuuden varmistamiseksi. Taulukossa 2-2 on listattu yleisimpiä radio-observatorioiden sisäisiä häiriölähteitä. Laitteiden vaikutukset radioastronomiseen mittauksiin riippuvat suuresti observatorion infrastruktuurista sekä käytettävissä olevan häiriösuojauksen tasosta. [1] [26]

Infrastruktuurin RFI	**Tieteellisten instrumenttien RFI**	**Muu ”sisäinen” RFI**
Tietokoneet ja niiden oheislaitteet: Näytöt, Ulkoiset kiintolevyt	Vetymaserit	Autojen bensiinimoottorit ja peruutustutkat
Langattomat yhteydet: Bluetooth, WLAN	Radioteleskooppia ohjaava servoelektroniikka	Mikroaaltouunit
Loisteputkivalot, LED-valojen virtalähteet	Spektrometrit, Signaaligeneraattorit, Digitaaliset mittalaitteet	Mobiililaitteet (matkapuhelimet)
Rakennuksen lämmitys ja ilmastointi (HVAC)	(Häiriömonitorit)	

Taulukko 2-2: Radio-observatorioiden sisäiset häiriölähteet [1, p. 17]

2.5.3 Häiriöiden vaikutus mittauksiin

Erilaiset häiriöt kytkeyvät mittauksiin eri tavoin. Vastaanotinantennin pää- ja sivukeilojen kautta kytkeytyvät signaalit voivat saturoida tai jopa rikkoa vastaanottimien sisältämiä vahvistimia, jos niiden tehotaso nousee yli 0,1 – 0,01 watin tasolle. Vastaanotinantennin sivukeilan läpi vuotavien häiriöiden tapauksessa kyseisen tehotaso saavutetaan Jukka-Pekka Porkon diplomityön mukaan vastaanottimesta riippuen vastaanotetun signaalin sähkövuon tiheyden ollessa välillä -10...+40 dBW/m^2. Tätä pienemmätkin häiriötehot voivat tosin aiheuttaa vastaanottimen vahvistimien kompressoitumista. Porkon mukaan arviot vastaanottimien yhden prosentin kompressoitumisen aiheuttavalle sähkövuon tiheydelle ovat luokkaa -70...-30 dBW/m^2, kun toimitaan 3 - 30 GHz:n välisellä taajuusalueella. Tarkempaa tietoa kyseisistä raja-arvoista voidaan löytää kansainvälisen televiestintäliiton suosituksesta ”ITU-R RA. 769-2 Protection criteria for radio astronomical measurements” [27]. [1] [3, pp. 32-35] [9]

Itse mittauksiin kyseiset virheet vaikuttavat seuraavalla tavalla. Jo yllä mainituilla vahvistimien kompressoitumista aiheuttavilla sähkövuon tiheyden tasoilla ajaudutaan vastaanottimen epälineaariselle alueelle, jolloin signaali säröytyy eli sen harmoniset komponentit vahvistuvat. Tämä leventää häiriösignaalin vaikutusaluetta, jolloin se voi peittää alleen tai vaimentaa itse mitattavan signaalin osia. Häiriösignaalin levenemisen ansiosta sen ei tarvitse osua edes lähelle mitattavan signaalin taajuusaluetta häiritäkseen mittauksia. Vahvistimien saturoituminen taas leikkaa niiden ulostulon maksimiarvoihinsa, jolloin vastaanotin ylikuormittuu ja kyseisen ajanhetken mittaustulokset menetetään. Radiotaajuiset häiriöt voivat edellä mainittujen kytkeytymistapojen lisäksi vääristää mittausten tuloksia kytkeytymällä suoraan näytteistetyn signaalin alasmuutetulle IF-välitaajuudelle. Tällöin häiriösignaalit näkyvät suoraan mittaustuloksissa, jolloin ne voivat heikentää mittausten luotettavuutta. [1] [28]

2.5.4 Häiriöiltä suojautuminen

Radioastronomisten vastaanottimien herkkyyden sekä radiohäiriöympäristön häiriölähteiden määrän kasvu lisäävät radioastronomisten mittausten häiriöiden suodattamisen tärkeyttä. Yleisimmin nykyään käytössä olevat mittausten suodatusmenetelmät perustuvat lähinnä datan visuaalisesti tarkasteltuna selviä häiriöitä sisältävien osien suoraan hylkäämiseen. Tämän tyyppiset menetelmät eivät kuitenkaan sovellu erityisen hyvin varsinkaan laajakaistaisten tai interferometristen mittausten suodattamiseen. Edistyneempiä suodatusmenetelmiä tarvitaankin tämän tyyppisissä mittauksissa varsinkin, jos mittaukset suoritetaan radioastronomisille mittauksille varattujen, suojattujen taajuuskaistojen ulkopuolella. [8]

Mittauksiin kytkeytyvien radiotaajuisten häiriöiden määrä riippuu suoritettavan mittauksen taajuusalueesta, ulkoisten häiriölähteiden lukumäärästä ja etäisyydestä mittauksiin, mittauksessa käytetyn integraatioajan pituudesta, mittausajankohdasta, mittauksen polarisaatiosta sekä mittaussuunnasta [1]. Radiohäiriöiden vaikutusta mittauksiin voidaankin vähentää seuraavilla menetelmillä: mittaamalla ja tunnistamalla haitallisia häiriöitä tuottavat, mittausten kannalta tarpeettomat häiriölähteet ja poistamalla ne käytöstä; suorittamalla mittaukset suojatuilla radioastronomisiin mittauksiin varatuilla taajuuskaistoilla; kehittämällä radioastronomiaan liittyvää regulatiivista toimintaa ja rajaamalla entistä leveämpiä kaistoja radioastronomian käyttöön; suodattamalla mitattua signaalia

analogisen suodatuksen ja/tai digitaalisen signaalinkäsittelyn avulla; sekä käyttämällä esimerkiksi interferometrisiä menetelmiä. [8] [25] [24]

Digitaaliset signaalinkäsittelyyn perustuvat suodatusmenetelmät ovat erittäin tutkittuja radioastronomian tieteenalalla niiden nopean kehittymisen sekä kustannustehokkuuden ansiosta. Radioastronomiassa käytettävät suodatusmenetelmät voivat perustua esimerkiksi mitatun tehon raja-arvon ylittävien signaalin osien hylkäämiseen aika- tai taajuustasossa (engl. time- and frequency blanking) tai adaptiivisiin ja spatiaalisiin suodatusmenetelmiin, joissa itse mitattavaa signaalia suodatetaan erillisen häiriömittauksen avulla esimerkiksi käyttämällä interferometrisiä menetelmiä ja mittausten keskinäistä korrelointia. Häiriöitä voidaan myös tunnistaa ja suodattaa tilastollisten menetelmien avulla. Esimerkiksi spektrin huipukkuuden (engl. Spectral Kurtosis, SK) mittaukseen perustuva menetelmä nähdään J-P Porkon diplomityön [1, pp. 25-31] mukaan niin tehokkaana työkaluna häiriösignaalien havaitsemiseen ja poistamiseen, että sen on arvioitu muodostuvan jopa standardiksi seuraavan sukupolven radioteleskoopeissa. [8]

Tietyt suodatusmenetelmät toimivat tehokkaasti tietyn tyyppisten häiriöiden suodattamisessa, eikä kaikkiin häiriötyyppeihin tehokasta yksittäistä suodatusmenetelmää ole olemassa. Kaikki häiriösuodatusmenetelmät perustuvat kuitenkin häiriöiden erottamiseen itse mitattavasta signaalista, mikä korostaa itse häiriöiden havaitsemisen tärkeyttä. Lisäksi häiriösignaalien tunteminen edesauttaa ja yksinkertaistaa entistä parempien suodatusalgoritmien kehitystä. [8]

Interferometrisillä menetelmillä voidaan parantaa mittausten vastustuskykyä radiotaajuisia häiriöitä vastaan, sillä erillisten radioteleskooppien vastaanottamat häiriöt eivät riipu toisistaan, jolloin ne suodattuvat tehokkaasti signaalien korrelointivaiheessa. Pitkäkantainterferometriassa (engl. Very Long Baseline Interferometry, VLBI) vastaanottolaitteiston edut häiriökestävyydessä korostuvat entisestään, sillä käytettävät radioteleskoopit sijaitsevat täysin toisistaan riippumattomissa häiriöympäristöissä eri puolilla maapalloa. VLBI-observaatioiden avulla onkin mahdollista saavuttaa noin 40 desibeliä korkeampi häiriönsiedon taso verrattuna vastaaviin yhdellä radioteleskoopilla suoritettaviin havaintomenetelmiin. [1] [29, pp. 7-18]

2.6 Häiriöiden mittaamisen perusperiaatteita

Normaalisti signaaleja mitattaessa toimitaan aikatasossa, jolloin mittaukset voidaan suorittaa esimerkiksi oskilloskoopilla. Häiriömonitoroinnissa mittaustulokset on kuitenkin hyödyllistä esittää taajuustasossa, sillä näin mahdollistetaan koko mitatun taajuusalueen tapahtumien samanaikainen tarkastelu. Spektristä voidaan esimerkiksi nähdä suoraan, paljonko energiaa mitatussa signaalissa on sen eri taajuuksilla, mikä selventää signaalin koostumuksen havainnointia. Taajuustason mittaukset suoritetaan useimmiten spektrianalysaattorien avulla, joskin nykypäiväiset oskilloskoopit ja signaalianalysaattorit sisältävät usein myös toimintoja mitattujen signaalien spektrin esittämiseen. Spektrianalysaattorit ja oskilloskoopit suunnitellaan kuitenkin lähtökohtaisesti eri käyttökohteisiin. Oskilloskoopeilla signaalin näytteistykseen käytetään yleensä mahdollisimman nopeita komponentteja, millä pyritään saavuttamaan hyvä aikatason resoluutio. Spektrianalysaattoreilla taas pyritään suurempaan dynaamiseen alueeseen, mikä mahdollistaa signaalin eri amplituditasojen tarkemman mittaamisen ja amplitudiltaan toisistaan eroavien,

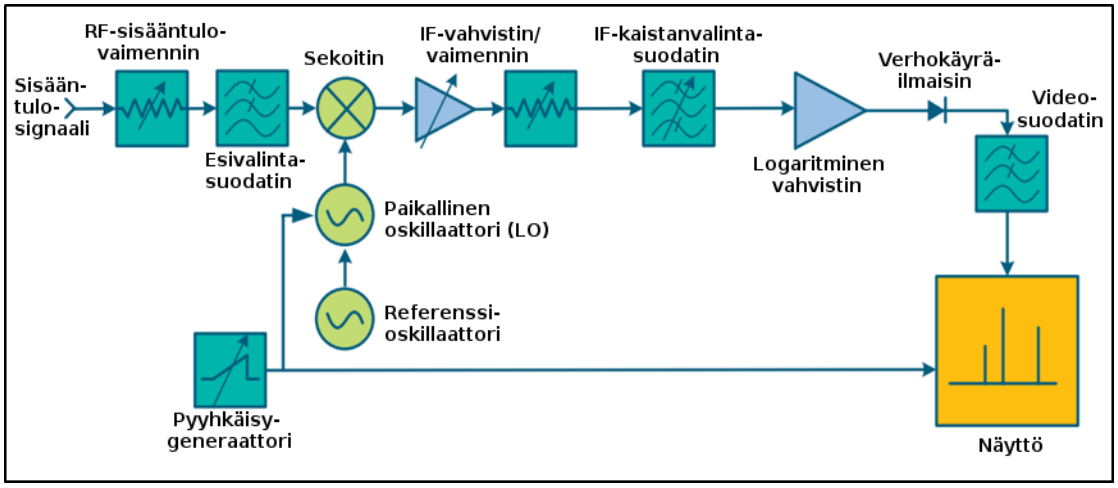
taajuustasossa lähekkäin sijaitsevien signaalien samanaikaisen havaitsemisen. [30] [31] [32, p. 10]

2.6.1 Spektrianalysaattorien eri tyypit ja toiminta

Spektrianalysaattorit voidaan jakaa pyyhkäiseviin-, FFT- ja hybridi -spektrianalysaattoreihin niiden rakenteen sekä toiminnan erojen suhteen. Pyyhkäisevät spektrianalysaattorit perustuvat viritettävään radiovastaanottimeen, jonka vastaanottotaajuutta muuttamalla on mahdollista näytteistää hyvin laajaa taajuusaluetta. Pyyhkäisevän spektrianalysaattorin sisältämien suodattimien kaistanleveys määrittää varsin suoraan sillä saavutettavan taajuustason resoluution. Toisaalta kapeat suodattimet ja etupään viritysviiveet heikentävät laitteella saavutettavaa aikatason resoluutiota. [30] [31]

FFT-spektrianalysaattorin toiminta perustuu Fourier'n muunnokseen. FFT-spektrianalysaattori voikin tuottaa taajuustason esityksen lähes reaaliajassa, eli sillä saavutetaan huomattavasti parempi aikatason resoluutio verrattuna pyyhkäisevään spektrianalysaattoriin. Sisääntulosignaalin suora näytteistys ilman erillistä radiovastaanotinta rajoittaa kuitenkin nykyisillä FFT-spektrianalysaattoreilla mitattavissa olevan taajuusalueen korkeintaan noin gigahertsin tasolle. [30] [31]

Hybridispektrianalysaattori on edellä kuvattujen spektrianalysaattorityyppien yhdistelmä, jossa radiovastaanottimen avulla alemmalle taajuudelle muunnettu taajuuskaista analysoidaan Fourier'n muunnosten avulla. Tämä mahdollistaa eri spektrianalysaattorityyppien vahvuuksien samanaikaisen hyödyntämisen. Tässä luvussa keskitytäänkin lähinnä pyyhkäiseviin ja hybridi –tyyppisiin, heterodyne–rakennetta hyödyntäviin spektrianalysaattoreihin. Kuvassa 2.2 on esitetty tämän tyyppisen spektrianalysaattorin yksinkertaistettu sisäinen rakenne. [30] [31]



Kuva 2.2: Heterodyne-tyyppisen spektrianalysaattorin yksinkertaistettu rakenne. Kuva muokattu lähteestä [30, p. 9]

Kuvasta 2.2 nähdään, että sisääntulosignaali etenee RF-sisääntulovaimentimen ja esivalintasuodattimen läpi sekoittajalle, jossa se sekoitetaan paikallisoskillaattorin (engl. Local Oscillator, LO) avulla tuotettuun signaaliin. RF-sisääntulovaimentimen avulla varmistetaan, että vastaanotettu signaali on sopivalla tasolla ennen sen kytkeytymistä

sekoittajalle. Järjestelyllä pyritään vähentämään sekoittajan ylikuormitusta, vahvistuksen kompressoitumista ja tästä johtuvaa signaalin säröytymistä. Sekoittaja on epälineaarinen laite, jonka ulostulona saadaan sen sisääntulosignaalit, näiden harmoniset moninkerrat sekä edellä mainittujen summa- ja erosignaalit. Sekoittajan avulla on siis mahdollista ”siirtää” signaaleja eri taajuuksille valitsemalla sen ulostulon signaaleista halutulla taajuudella olevat signaalin osat IF-kaistanvalintasuodattimen avulla. Sekoittajan ulostuloon saatava signaali vahvistetaan tämän jälkeen IF-vahvistimella, joka kompensoi RF-vaimentimen, mikserin sekä kaistanvalintasuodattimen aiheuttamaa vaimennusta. Näin näytteistettävän signaalin taso saadaan muunnettua vastaamaan sisääntulevan signaalin tasoa. Mitatut signaalit muokataan tämän jälkeen esitettävään muotoon logaritmisen vahvistimen, verhokäyräilmaisimen ja videosuodattimen avulla. Pyyhkäisygeneraattorilla tuotettu signaali ohjaa paikallisen oskillaattorin sekä näytön x-akselin toimintaa, jolloin tietty näytön x-akselin arvo vastaa tiettyä taajuutta. Spektrianalysaattorilla mitatut signaalit esitetäänkin useimmiten ”X-Y”–kuvaajassa, jossa x-akseli esittää taajuuden muutosta ja y-akseli signaalin amplitudia. Signaalit voidaan esittää esimerkiksi lineaariasteikolla voltteina tai logaritmisellä asteikolla desibeleinä. [30] [31]

2.6.2 Heterodyne- tyyppisten spektrianalysaattorien mittauskaista

Heterodyne –periaatteella toimivan pyyhkäisevän tai hybridi- tyyppisen spektrianalysaattorin hetkellinen mittauskaista riippuu käytetyn IF-kaistanpäästösuodattimen keskitaajuudesta, kaistanleveydestä sekä sekoittimen virittämiseen käytetyn LO:n viritystaajuudesta. Paikallisen oskillaattorin ja sekoittimeen kytkeytyneiden ulkopuolisten häiriösignaalien taajuus ei saisi osua mitattavalle taajuuskaistalle, sillä muutoin ne kytkeytyvät IF-kaistanpäästösuodattimen läpi aiheuttaen mittauksiin huomattavia virheitä. Ulkopuolisten signaalien pääsy sekoittajalle tulisi siis estää ja toisaalta paikallisen oskillaattorin taajuus tulisi valita mittausalueen ulkopuolelta. Tällöin systeemin viritysfunktio voidaan määritellä kaavassa 2.1 esitetyllä tavalla: [30, p. 11] [31] [32]

$$f_{sig} = |f_{LO} \pm f_{IF}|, \tag{2.1}$$

jossa f_{sig} ja f_{LO} ovat signaalin ja paikallisen oskillaattorin taajuudet ja f_{IF} on IF-kaistanpäästösuodattimen keskitaajuus eli mittauskaistan IF-välitaajuus. Yleensä signaalit halutaan siirtää pienemmälle taajuudelle, jolloin tarvittava paikallisen oskillaattorin taajuus voidaan ratkaista yhtälöllä 2.2. [30, p. 11]

$$f_{LO} = f_{sig} - f_{IF} \tag{2.2}$$

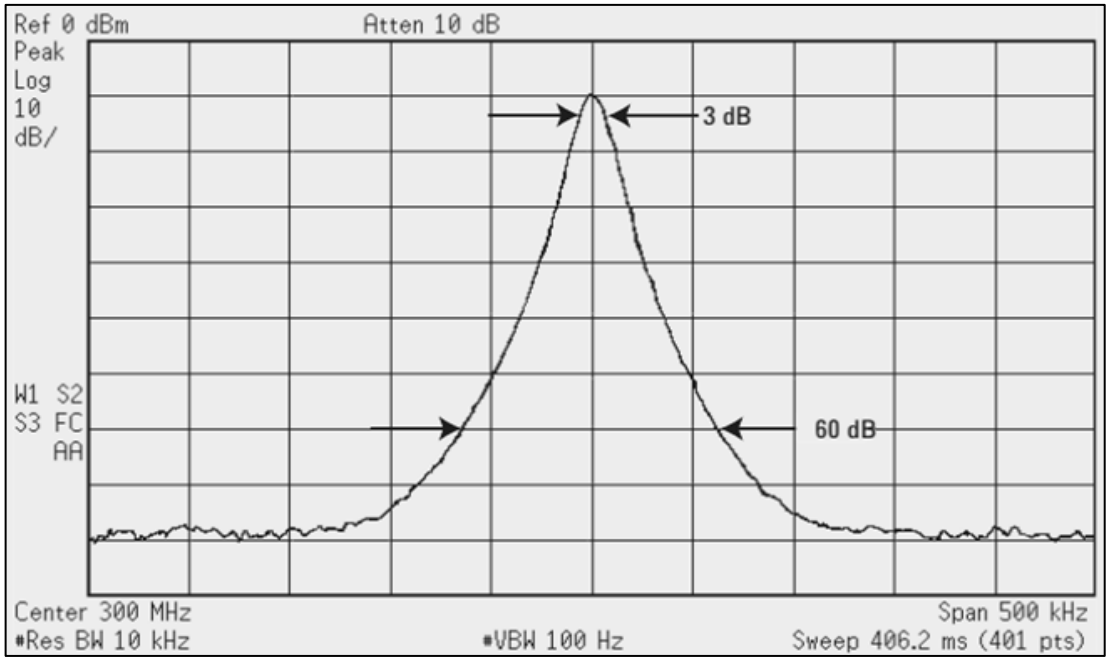
2.6.3 Spektrianalysaattorien taajuustason resoluutio ja selektiivisyys

Spektrianalysaattorin resoluutiolla tarkoitetaan sen kykyä erottaa taajuustasossa lähekkäisiä signaaleja toisistaan. Teoriassa puhtaat sinisignaalit näkyisivät spektrissä yksittäisinä viivoina, jolloin niiden havaitseminen olisi hyvin yksinkertaista. Todellisuudessa laitteen epäideaalisuudet, kuten suodatinten rajallinen vaimennus, sekoittimien aiheuttamat ylimääräiset signaalit, vahvistinten epälineaarisuus sekä oskillaattorien taajuuden ja vaiheen epävarmuus johtavat signaalien taajuustason vasteen levenemiseen. Spektrianalysaattorin resoluutiokaistanleveyttä (engl. resolution bandwidth, RBW) kaventamalla voidaan kuitenkin edellä mainittujen epäideaalisuuksien rajoissa vähentää signaalien levenemistä, mikä mahdollistaa taajuustasossa lähekkäisten signaalien erottamisen toi-

sistaan. Pyyhkäisevällä spektrianalysaattorilla saavutettava taajuustason resoluutio Δf määräytyy suoraan valitun IF-kaistanvalintasuodattimen kaistanleveydestä, kun taas FFT- ja hybridispektrianalysaattoreilla se määräytyy näytteenottoon käytetyn ajan t_{AQT} tai käytetyn näytteenottotaajuuden F_S ja suoritettavan FFT:n pituuden FFT_{len} funktiona kaavassa 2.3 esitetyllä tavalla. [30] [32, pp. 51-53] [33, p. 13]

$$\Delta f = RBW = \frac{1}{t_{AQT}} = \frac{F_S}{FFT_{len}} \tag{2.3}$$

Saavutettava taajuustason resoluutio ei kuitenkaan suoraan määrittele järjestelmällä saavutettavaa lähekkäisten signaalien havaittavuutta, sillä mitattujen signaalien vaste riippuu myös käytetystä kaistanvalintasuodattimesta. Esimerkiksi Agilent käyttää datalehdissään 3dB:n kaistanleveyden arvoa kaistanvalintasuodattimilleen. Tämä arvo kertoo signaalien taajuuden eron, jolla amplitudiltaan yhtäsuurten signaalien välissä nähdään vielä suuruusluokaltaan 3 dB:n kuoppa niiden taajuustason esityksessä. Usein signaalit eivät kuitenkaan ole amplitudiltaan yhtä suuria, jolloin niistä amplitudiltaan pienempi voi hävitä suurempiamplitudisen signaalin aiheuttaman ”helman” alle. Signaalin aiheuttaman helman leveyttä kuvataan Agilent:n datalehdissä kaistanleveyden selektiivisyydellä, joka on signaalin taajuustason esityksen -3 dB:n ja -60 dB:n kaistanleveyksiä vertaava suhdeluku. Kuvassa 2.3 on esitetty kyseisen suhdeluvun muodostamista selventävä esimerkki, jossa neljäasteisella kaistanvalintasuodattimella on saatu kaistanleveyden selektiivisyyden suhdeluvuksi 12,7:1. Tällöin siis signaalista mitattu -60 dB:n kaistanleveys on 12,7-kertainen verrattuna -3 dB:n kaistanleveyteen. [30, p. 14] [32, pp. 51-53]



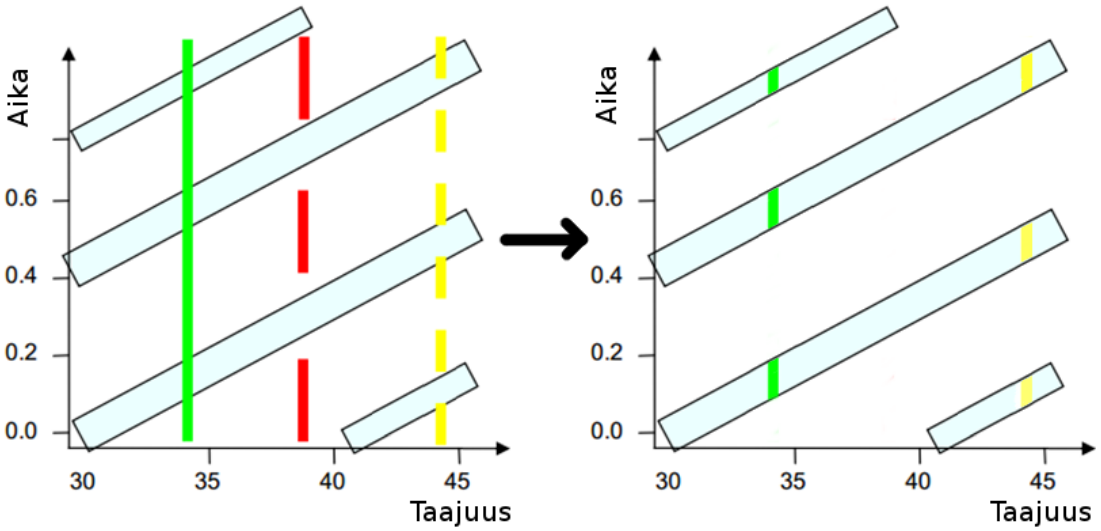
Kuva 2.3: Taajuuskaistan selektiivisyys eli signaalin -3dB ja -60dB kaistanleveyksien suhde. Kuva muokattu lähteestä [30, p. 14]

2.6.4 Spektrianalysaattorien pyyhkäisyaika ja aikatason resoluutio

Jos taajuustason resoluutio olisi spektrianalysaattorin ainut hyödyllinen ominaisuus, suunniteltaisiin kaikki spektrianalysaattorit käyttäen mahdollisimman ohuita kaistanvalintasuodattimia. Pyyhkäisevällä spektrianalysaattorilla saavutettava pyyhkäisynopeus riippuu kuitenkin RBW:n kaistanleveydestä. Kaistanvalintasuodattimet ovat kaistarajoitettuja elektronisia piirejä, jotka tarvitsevat aikaa niiden sisältämien komponenttien varautumiseen sekä varauksen purkautumiseen. Liian nopean pyyhkäisyn käyttäminen vaimentaakin kaistanvalintasuodattimen läpi kytkeytyvien signaalien amplitudeja huomattavasti. Lisäksi signaalien paikka siirtyy taajuustasossa. Pyyhkäisevän spektrianalysaattorin virheettömään pyyhkäisyyn tarvittavaa aikaa voidaankin arvioida yhtälön 2.4 avulla: [30, p. 18]

$$t_{sweep} = \frac{k(B_{span})}{RBW^2} \tag{2.4}$$

jossa t_{sweep} on taajuuskaistan pyyhkäisyyn tarvittava aika, B_{span} on pyyhkäistävän taajuuskaistan leveys ja k on käytetystä kaistanvalintasuodattimesta riippuva vakio. Pyyhkäisyyn tarvittava aika ja käytetty resoluutiokaistanleveys vaikuttavat suoraan pyyhkäisevällä spektrianalysaattorilla saavutettavaan aikatason resoluutioon, sillä yksittäinen, tietyllä taajuudella oleva signaali kytkeytyy laitteen IF-kaistanvalintasuodattimen läpi vain hyvin lyhyen ajan koko taajuusalueen pyyhkäisyn aikana. Kuva 2.4 selventää edellä kuvattua tilannetta. Kuvassa keltainen signaali havaitaan jatkuvana signaalina vihreän signaalin tapaan, kun taas punaista signaalia ei havaita ollenkaan. RBW:n rajaaman mittauskaistan täytyykin osua mitattavan signaalin kaistalle oikealla ajanhetkellä, jotta kyseinen signaali, tai sen puuttuminen, voidaan havaita. [30] [32] [34]



Kuva 2.4: Pyyhkäisevän spektrianalysaattorin aikatason resoluutio [34, p. 12]

FFT- ja hybridispektrianalysaattoreilla näytteistetään useimmiten pyyhkäiseviin spektrianalysaattoreihin nähden hyvin leveää hetkellistä taajuuskaistaa ja taajuustason resoluutio luodaan mittaustuloksista jälkikäteen Fourier'n muunnoksella. Fourier'n muunnos voi vastata operaationa jopa satojen rinnakkaisten RBW-suodattimien käyttöä, minkä ansiosta suodattimien asettumisaikojen vaikutus saadaan minimoitua. Pyyhkäisyyn ku-

luva aika kasvaakin hybridispektrianalysaattoreilla vain lineaarisesti RBW:n käänteisarvon funktiona kaavassa 2.4 esitetyn RBW:n käänteisarvon neliön sijaan. Hybridispektrianalysaattorilla saavutetaankin varsin suuria nopeusetuja pyyhkäisevään spektrianalysaattoriin verrattuna, kun käytössä on pieni RBW:n arvo ja näytteistetään suhteellisen leveää taajuusaluetta. Erittäin laajoilla kaistanleveyksillä FFT:n laskennallinen monimutkaisuus voi tosin johtaa saavutetun nopeusedun menettämiseen. [35]

Hybridispektrianalysaattorien pyyhkäisyaikaa voidaan arvioida kaavassa 2.5 esitetyllä tavalla: [35]

$$t_{sweep} = (N_{FFT}) \times \left(t_{AQT} + t_{PROS} + t_{LO}\right), \tag{2.5}$$

jossa N_{FFT} on taajuusalueen näytteistämiseen tarvittavien viritysaskeleiden lukumäärä, t_{AQT} ja t_{PROS} ovat yksittäisen FFT:n näytteiden näytteistämiseen- sekä prosessointiin tarvittavat ajat, kun taas t_{LO} kuvaa laitteen etupään yksittäiseen viritykseen tarvittavaa aikaa. Yksittäisen FFT:n näytteiden näytteistämiseen kuluvaa aikaa voidaan arvioida kaavassa 2.3 esitetyllä tavalla, eli t_{AQT} riippuu käytetystä näytteistysnopeudesta sekä suoritettavan FFT:n pituudesta. Alarajan näytteistysajalle asettaa kuitenkin käytetyn kaistanvalintasuodattimen asettumisaika. Erittäin nopeilla näytteistysnopeuksilla tai lyhyillä FFT:n pituuksilla toimittaessa, näytteiden näytteistämiseen tarvittavaa minimiaikaa voidaankin siis arvioida kaavalla 2.6, jossa k on käytetystä kaistanvalintasuodattimesta riippuva vakio. [35]

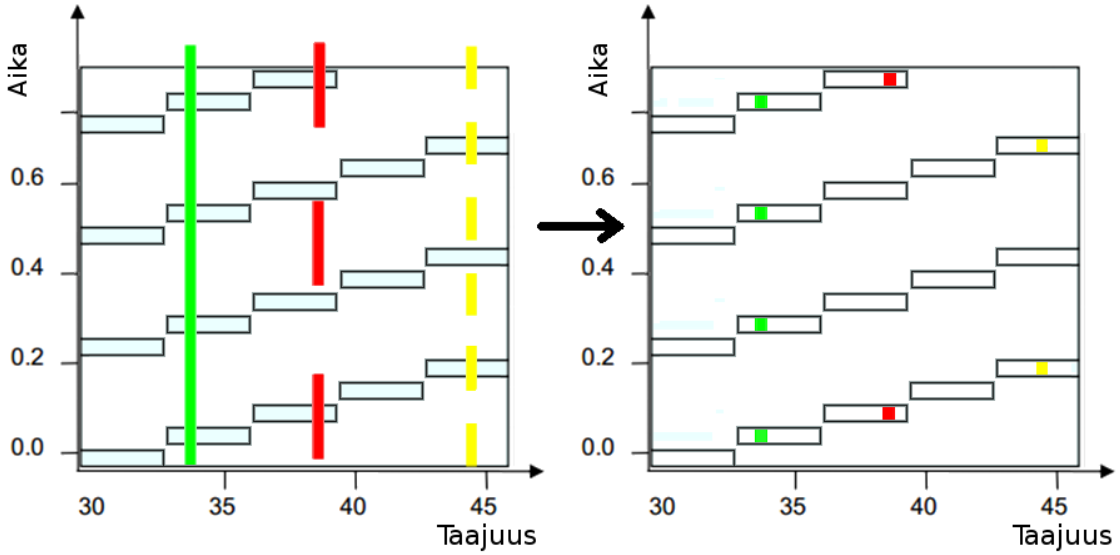
$$t_{AQT_{min}} = \frac{k}{RBW} \tag{2.6}$$

Jos suoritettavan Fourier'n muunnoksen laskemiseen kuluu vähemmän aikaa kuin itse muunnoksen näytteiden näytteistämiseen, voidaan prosessit suorittaa rinnakkain, jolloin kaavassa 2.5 esitetty pyyhkäisyaika voidaan supistaa kaavassa 2.7 esitettyyn muotoon. [35]

$$t_{sweep} = (N_{FFT}) \times \left(t_{AQT} + t_{LO}\right) \tag{2.7}$$

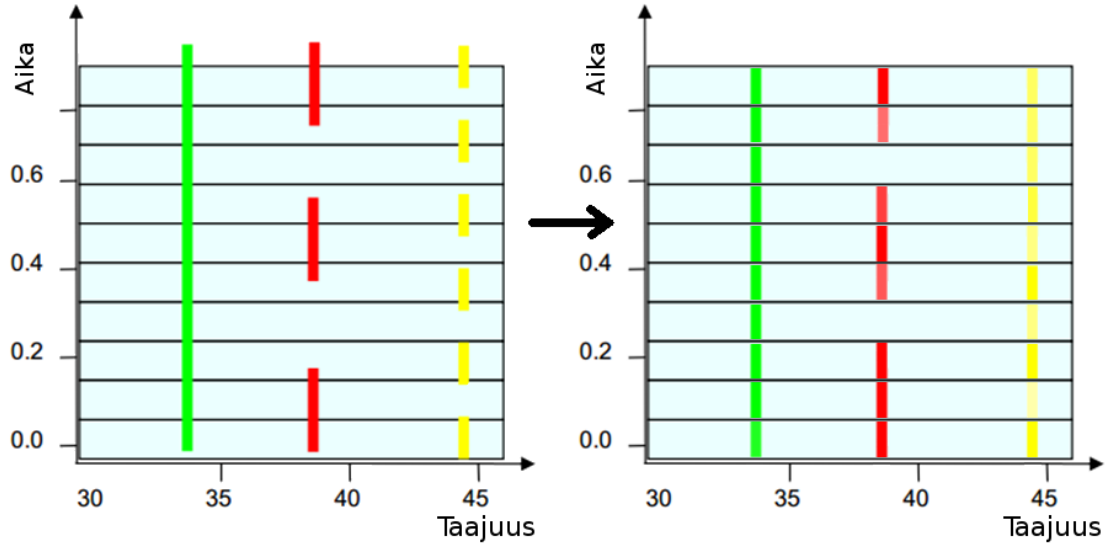
Käytetyn laitteiston arkkitehtuuri ja prosessointiteho vaikuttavat FFT:n laskemiseen tarvittavaan aikaan. Pyyhkäisyaikaan vaikuttaa myös näytteistettävä hetkellinen kaistanleveys, sillä se vaikuttaa tarvittavien viritysaskeleiden lukumäärään. Monimutkaisuudesta johtuen tämän tyyppisillä spektrianalysaattoreilla saavutettavan pyyhkäisyn nopeuden arviointi onkin vaikeaa. Lisäksi joidenkin valmistajien laitteen eivät ota huomioon liian nopeasta pyyhkäisystä johtuvia virheitä, mistä johtuen datalehdissä kuvatut pyyhkäisyajat voivat antaa liian optimistisen kuvan laitteen suorituskyvystä. [35]

Kuvassa 2.5 on havainnollistettu hybridispektrianalysaattorin avulla saavutettavaa aikatason resoluutiota. Pyyhkäisevää spektrianalysaattoria laajempi hetkellinen näytteistettävä kaistanleveys mahdollistaa useiden lähekkäisten signaalien samanaikaisen havaitsemisen. Lisäksi se mahdollistaa pyyhkäisyn nopean suorittamisen, mikä parantaa järjestelmällä saavutettavaa aikatason resoluutiota. [35]



Kuva 2.5: Hybridispektrianalysaattorin aikatason resoluutio [34, p. 12]

Signaalien näytteistyksen ja FFT:n laskemisen rinnakkaistaminen voi parhaimmillaan mahdollistaa taajuustason esityksen luomisen reaaliajassa, jos laitteen etupäätä ei viritetä mittausten välillä. Tällä tavalla toimivaa spektrianalysaattoria kutsutaankin reaaliaikaiseksi FFT-spektrianalysaattoriksi. Yksittäisten signaalien havaitsemisessa kyseinen reaaliaikaisuus tarkoittaa sitä, että tietty näytteistettävälle kaistalle osuva signaali havaitaan aina, eli näytteistysikkunoiden väliin ei jää tyhjää tilaa aikatasossa. FFT:n tarvitsemien näytteiden näytteistämiseen tarvittava aika (kaavat 2.3 ja 2.6) rajoittaa kuitenkin laitteella saavutettavaa aikatason resoluutiota. Tilannetta on selvennetty kuvassa 2.6, jossa kaikki kolme signaalia havaitaan taajuustason kannalta täysin virheettömästi, mutta niiden aikatason selkeitä rajakohtia ei nähdä kuin tietyillä rajoitetuilla ajanhetkillä. [33] [36]



Kuva 2.6: Reaaliaikaisen FFT-spektrianalysaattorin aikatason resoluutio [34, p. 12]

2.6.5 Herkkyys (DANL-arvo ja kohinaluku)

Herkkyys, eli laitteen kyky havaita pieniamplitudisia signaaleja, on eräs spektrianalysaattorien tärkeimmistä parametreistä. Saavutettavissa olevaa herkkyyttä rajoittaa laitteen itsensä tuottama kohina, joka syntyy sen komponenttien sisäisestä satunnaisesta elektroniliikkeestä. Spektrianalysaattorin kohinalla viitataan yleensä laitteen näytettyyn keskiarvoiseen kohinatasoon (engl. Displayed Average Noise Level, DANL), joka on laitteen termisen kohinan ja kohinaluvun summa. DANL voidaan määrittää suoraan spektrianalysaattorin näytöllä näkyvästä kohinatasosta, kun laitteen sisääntulo on terminoitu 50 ohmin päätteellä. Päätteen tuottama kohinateho saadaan kaavalla 2.8: [30]

$$P_{n(dBW)} = 10\ \log(kTB), \tag{2.8}$$

jossa k on Boltzmann:n vakio eli 1,38 x 10^{-23} J/K, T on päätteen lämpötila kelvinasteissa ja B on kaistanleveys hertseissä. Kohinateho normalisoidaan yleensä 1 Hz:n kaistanleveyteen ja sen referenssitehona käytetään milliwattia (dBm). Tällöin kaava 2.8 voidaan muuntaa kaavassa 2.9 nähtävään muotoon. [30, p. 46]

$$P_{n(dBm)} = 10\ \log(kTB/1\ mW) \tag{2.9}$$

Kaavan 2.9 avulla saadaan päätteen huoneenlämpöiselle (290K) kohinatehotiheydelle arvo -174 dBm/Hz. Päätteen generoima kohina vahvistuu vielä spektrianalysaattorin vahvistimissa, jotka lisäävät siihen itse tuottamansa kohinan. Sisääntulovaimennin, sekoitin ja muut laitteen signaalitiellä ennen ensimmäistä vahvistinta sijaitsevat osat lisäävät kohinaa vain vähän. Ne kuitenkin vaimentavat signaalia, mikä heikentää laitteen herkkyyttä, laitteella saavutettavaa signaalin ja kohinan suhdetta (engl. Signal-to-Noise Ratio, SNR) sekä laitteen DANL-arvoa. DANL-arvon voidaankin siis ajatella kuvaavan spektrianalysaattorin ”tehollista kohinatasoa”, sillä se ottaa huomioon myös signaalitiellä tapahtuvan vaimennuksen. [30] [32]

Valittu RBW vaikuttaa myös laitteella saavutettavaan herkkyyteen. Spektrianalysaattorin generoima kohina on sattumanvaraista ja varsin leveäkaistaista. RBW-suodattimet sijaitsevat signaalitiellä IF-vahvistinten jälkeen ja niiden kaistanleveys vaikuttaa suoraan näytölle kytkeytyvän kokonaiskohinatehon määrään. RBW:n muuttamisesta johtuvaa näytöllä näkyvää kohinatason vaihtelua ΔN_f voidaan kuvata kaavan 2.10 avulla: [30, p. 47]

$$\Delta N_f = 10\log(RBW_2/RBW_1), \tag{2.10}$$

jossa RBW_1 ja RBW_2 ovat vertailtavat resoluutiokaistanleveydet. RBW:n muuttaminen kertoimella kymmenen muuttaa siis myös näytöllä näkyvää kohinatasoa kymmenellä desibelillä. Paras SNR ja suurin herkkyys (pienin DANL-arvo) saavutetaankin käyttämällä pienintä valittavissa olevaa sisääntulovaimennuksen- sekä RBW:n arvoa. Näin saatu DANL-arvo vastaa pienintä spektrianalysaattorilla suoraan mittaamalla havaittavissa olevaa signaalitasoa. [30] [32]

FFT- ja hybridispektrianalysaattorien RBW on käytetyn FFT:n pituuden sekä näytteistettävän mittauskaistan leveyden, eli näytteistysnopeuden funktio (kaava 2.3). Toisaalta

tämä siis kaavan 2.10 seurauksena tarkoittaa sitä, että käytetyn näytteistysnopeuden pienentäminen ja FFT:n pituuden lisääminen parantavat laitteella saavutettavaa dynaamista aluetta ja herkkyyttä, joskin samalla mittauksiin tarvittava pyyhkäisyaika pitenee. Pidempää FFT:tä käyttämällä saavutettavan herkkyyden kasvun ja pyyhkäisyajan suhde on kuitenkin pienillä FFT:n arvoilla vahvasti epälineaarinen, mikä mahdollistaa optimaalisten asetusten valinnan laitteelta vaadittavan herkkyyden, saavutettavan pyyhkäisyajan sekä aikatason resoluution suhteen. [35]

2.6.6 Herkkyyden lisääminen esivahvistimella

Sopivan esivahvistimen avulla on mahdollista parantaa mittajärjestelmän herkkyyttä ja kohinalukua entisestään. Esivahvistin siirtää spektrianalysaattorin mitta-aluetta vahvistuksensa verran, jolloin pienin havaittavissa oleva signaalitaso pienenee, joskin samalla pienennetään myös suurinta havaittavissa olevaa signaalitasoa. Toisaalta esivahvistin vahvistaa myös kohinaa, jolloin herkkyyden kasvattaminen voi kutistaa käytettävissä olevan mitta-alueen kokoa eli dynaamista aluetta. Paras absoluuttinen herkkyys saavutetaan käyttämällä esivahvistinta, jolla on korkea vahvistus ja pieni kohinaluku. Jos kuitenkin halutaan säilyttää koko mitta-alue, joudutaan valitsemaan vahvistin, jonka vahvistuksen ja kohinaluvun summa on vähintään kymmenen desibeliä pienempi kuin spektrianalysaattorin kohinaluku. Tällöin kokonaissysteemin kohinapohja laskee esivahvistimen vahvistuksen verran. Sisääntulovaimenninta voidaan käyttää esivahvistimen vahvistuksen pienentämiseen ja näin edesauttaa edellä kuvatun raja-arvon täyttymistä. Monet spektrianalysaattorit sisältävät sisäänrakennettuja esivahvistimia, mikä mahdollistaa järjestelmän kalibroinnin ja herkkyyden maksimoinnin. Käytettävän esivahvistimen optimaaliseen valintaan on löydettävissä neuvoja Agilent:n sovellusohjeesta ”Spectrum Analysis Basics (Application Note 150)” [30]. [32]

Luku 3

3 Metsähovin mittausympäristö

Tässä luvussa käydään läpi Metsähovissa radioastronomian tutkimuksessa käytettävän laitteiston sekä Metsähovin mittausympäristön pääpiirteitä. Lisäksi luvussa tarkastellaan Metsähovin radioympäristössä esiintyviä häiriölähteitä sekä niiden tuottamien häiriösignaalien aiheuttamien ongelmien minimointiin ja ennaltaehkäisyyn käytettäviä menetelmiä. Lisäksi luvun alussa käydään pintapuolisesti läpi Metsähovin observatorioalueen tutkimustoimintaa.

Kuva 3.1: Metsähovin radio-observatorio. Kuva muokattu lähteestä [37]

3.1 Metsähovin radiotutkimusasema

Metsähovin radiotutkimusasema on Aalto-yliopiston Sähkötekniikan korkeakouluun kuuluva laitos, joka sijaitsee Kirkkonummella Kylmälän kylässä noin 45 km länteen Helsingin keskustasta. Metsähovia lähimpänä sijaitsevan kyläkeskuksen Veikkolan keskusta sijaitsee noin 10 kilometrin etäisyydellä itse radio-observatoriosta. Metsähovin radio-observatoriota ympäröi säteeltään noin 550 metrin kokoinen radiohiljainen alue [38, p. 4]. Metsähovin radiotutkimusasema on ollut toiminnassa vuodesta 1974 ja erillislaitos se on ollut toukokuusta 1988 lähtien. Metsähovin radio-observatorion läheisyydessä toimii myös Helsingin yliopiston optiseen astronomian tutkimukseen keskittyvä observatorio sekä maa- ja metsätalousministeriön alaisuudessa toimiva avaruusgeodeettinen asema ja painovoimalaboratorio. [39]

Metsähovin radio-observatorion tehtävänä on tuottaa tutkimustyötä radiotieteen, radioastronomian ja avaruustekniikan aloilla, sekä edesauttaa radioastronomiassa käytettävien mikroaaltovastaanottimien, vastaanottotekniikoiden, antenni- ja tiedonkeruutekniikan, tiedon prosessoinnin sekä tieteellisen laskennan kehitystä. Suurin osa Metsähovin radio-observatoriossa tehtävästä tutkimuksesta keskittyy millimetri- ja mikroaaltotaa-

juuksille noin 100 MHz – 150 GHz taajuusalueelle. Observatorion tärkeimpiä tutkimuskohteita ovat Auringon millimetri- ja mikroaaltosäteily, radiokvasaarit ja kontinuumihavainnot, aktiiviset galaksit, molekyylien spektriviivat, geodeettinen ja radiotähtitieteellinen pitkäkantainterferometria sekä aktiivisten galaksien etuala-komponenttien kartoitus kosmisen taustasäteilyn tutkimusta varten. Metsähovi tukee myös radioastronomian tutkimusta ja sovellusten kehittämistä Suomessa yleisesti, lähinnä järjestämällä mahdollisuuksia opinnäytetöiden suorittamiseen ja jatko-opiskeluun (lisensiaatin- ja tohtorintutkinnot). [39]

3.2 Metsähovin mittalaitteisto

Metsähovissa radioastronomisissa mittauksissa käytetään vastaanotinantennina ensisijaisesti observatorion omaa, kuvassa 3.2 esitettyä, halkaisijaltaan 13,7 metristä Cassegrain-teleskooppia. Cassegrain-teleskooppi on yleisin radioastronomiassa käytetty teleskooppityyppi ja se koostuu parabolisesta pääpeilistä, kuperasta apupeilistä sekä vastaanotinmittalaitteistosta. Antenni heijastaa kerätyn säteilyn pääpeilin avulla apupeilille, josta se tarkentuu pääpeilin keskellä sijaitsevan aukon kautta vastaanotinmittalaitteistolle. Pääasialliset, Suomen oloissa helposti käytettävät havaintotaajuudet ovat 22 ja 37 GHz. Lisäksi käytetään taajuuksia 90 ja 150 GHz. Teleskoopin vastaanottimet toimivat lähinnä näillä suojatuilla taajuuksilla tuottaen ympärivuorokautista havaintodataa mm. kvasaari- ja aurinkotutkimuksen käyttöön. [1] [16] [40]

Kuva 3.2: Metsähovissa käytössä olevat vastaanotinantennit. Kuvassa vasemmalla 13,7 metrin Cassegrain-teleskooppi sekä sitä sään vaikutuksilta suojaava suojakupu ja oikealla 1,8 metrin ”Sunant”-vastaanotinantenni. Kuvat muokattu lähteistä [4] ja [40]

Metsähovin Cassegrain-teleskooppi on Suomen ainoa radioastronomisiin ja geodeettisiin mittauksiin käytettävä radioteleskooppi. Kyseinen radioteleskooppi otettiin käyttöön vuonna 1974 eli samana vuonna kuin itse radio-observatoriokin aloitti toimintansa. Radioteleskoopin pintapeilit sekä sitä suojaava suojakupu päivitettiin uudempiin vuo-

sien 1992-1994 aikana. Nykyisten käytössä olevien pintapeilien avulla saavutettu radioteleskoopin pintatarkkuus on 0,1 millimetriä, mikä mahdollistaa korkeintaan 150 GHz:n signaalien vastaanottamisen. Antennin suuntaamisen maksiminopeus on 1,2 astetta sekunnissa. [37, p. 19] [39]

Pienempää 1.8 metristä vastaanotinantennia (kuva 3.2) käytetään Metsähovissa Aurinkotutkimuksessa Auringon kokonaisintensiteetin seurantaan 11.2 GHz:n taajuusalueella. Myös kyseinen antenni on käytössä kokoaikaisesti lukuunottamatta huoltotoimenpiteistä aiheutuvia katkoja. Suuremman 13,7 metrisen radioteleskoopin mittausajasta on varattu yleensä noin 20 - 25% Aurinkotutkimuksen käyttöön, mikä mahdollistaa pienemmällä antennilla löydettyjen ilmiöiden tarkemman tarkastelun. [4] [41, pp. 29-30]

Metsähovissa on tämän työn kirjoituksen aikaan käytössä viisi radioastronomista vastaanotinta, joiden signaalitaajuudet, välitaajuudet sekä välitaajuuksilla mittausten kannalta häiriölähteinä toimivat muut käyttökohteet on esitetty taulukossa 3-1.

Vastaanotin	**Signaalin taajuus [GHz]**	**välitaajuus [GHz]**	**Muut välitaajuusalueen käyttäjät**
Geo-VLBI	2,21 - 2,35 8,15 - 8,65	0,68 - 0,82 0,5 - 0,98	Radiolinkit, langattomat kamerajärjestelmät, DVB
22 GHz Kontinuumi	21,0 - 22,0 22,4 - 23,4	0,2 - 1,2	Satelliittiyhteydet, lyhyen kantaman yhteydet, radiolinkit, DVB
22 GHz VLBI	21,98 - 22,48	0,5 - 1,0	Lyhyen kantaman yhteydet, radiolinkit, DVB
37 GHz Kontinuumi/ Aurinko	35,3 - 36,3 37,3 - 38,3	0,5 - 1,5	Tutkat, radiolinkit, DVB
37 GHz Aurinko	37,0	0,25 - 1,0	Radiolinkit, radionavigaatio, DVB
43 GHz VLBI	40,0 - 45,0	0,5 - 1,0 8,5 - 11,5	Lyhyen kantaman yhteydet, radiolinkit, DVB
80-115 GHz Spektriviiva	80 - 115	1,0 - 1,6	Radionavigaatio, satelliittinavigaatio
(90 GHz) 150 GHz “uusi 3mm” VLBI	84 - 89	*	
(90 GHz) 150 GHz VLBI	(84 - 115) 129 - 175	3,7 - 4,2 0,5 - 1,0	Radionavigaatio Radiolinkit, DVB
Callisto (Aurinko)	0,1 - 1,45		Radiolinkit, langattomat kamerajärjestelmät, DVB
RFI-monitori	0,4 - 2,0 (0,05 - 4,4)	(0,0 - 0,1)	Radionavigaatio, Radiolinkit

Taulukko 3-1: Radioastronomiset vastaanottimet Metsähovin radiotutkimusasemalla. Tähdellä merkittyjä kohtia ei ole vielä määritelty työn kirjoituksen aikaan. [4] [6] [42]

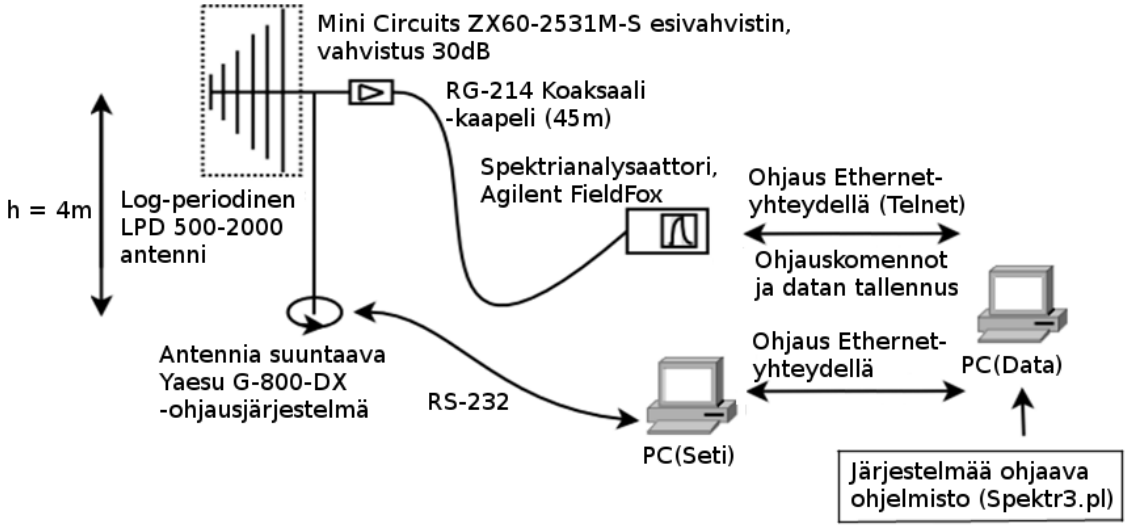
Metsähovin aurinkohavaintoihin käytettävää laitteistoa laajennettiin vuonna 2010 Callisto-mittausjärjestelmällä, kun Metsähovin radiotutkimusasema liittyi maailmanlaajuiseen ”e-Callisto”-mittausverkkoon. Metsähovin nykyinen Callisto-järjestelmä koostuu kahdesta ”Sunant”-vastaanotinantenniin kiinnitetystä log-periodisesta antennista, kahdesta Callisto-spektrometristä sekä edellä mainittujen oheiskomponenteista. Kuvassa 3.2 on nähtävissä järjestelmän vanha, yhteen log-periodiseen antenniin perustuva kokoonpano. Järjestelmän avulla on mahdollista suorittaa aurinkohavaintoja 100 - 1450 MHz:n taajuusalueella. [4] [41, p. 13] [43, pp. 27-28]

Metsähovin mittausten aikadatan synkronointiin käytetään kahden aktiivisen CH1 75 vety-maserin avulla luotua 5 MHz:n taajuustarkkaa vaihereferenssisignaalia, jota tarvitaan erityisesti VLBI-mittauksissa signaalien korreloinnin mahdollistamiseksi. Kyseistä signaalia käytetään myös vastaanotinten, spekrianalysaattorien ja signaaligeneraattorien paikallisten oskillaattorien vaihelukitukseen. Lisäksi vety-maserin avulla luodaan 1PPS-signaali, jota käytetään mittausten ajoittamiseen. Järjestelmän avulla voidaan varmistaa mitattujen signaalien ajoituksen, taajuuden ja vaiheen tarkkuuden pysyminen astronomisten mittausten vaatimalla tasolla. Vety-maserin toiminta perustuu vetyatomien sisäisten tunnettujen energiatasojen välisen 1420 450 751,768 hertsin oskillaation hyödyntämiseen. Kyseinen taajuus vastaa vedyn spektrissä näkyvää 21 cm:n spektriviivaa. Järjestelmän avulla voidaan luoda erittäin tarkka ja stabiili kelloreferenssi, sillä CH1 75 vety-maserin pitkäaikainen viiden vuoden taajuuden stabiilisuus on luokkaa $\pm 5 \times 10^{-16}$. [4] [37, p. 22]

3.3 Häiriömonitorointi Metsähovissa

Metsähovin radiotaajuista häiriöympäristöä on monitoroitu vuodesta 1998 lähtien vertikaalisella polarisaatiolla 400 - 2000 MHz:n taajuusalueella. Järjestelmää on tosin päivitetty useampaan kertaan, mikä vaikeuttaa eri vuosina suoritettujen häiriömittausten keskinäistä vertailua. Vuonna 2010 radio-observatorioon asennettiin uusi ohjattava RFI-monitori, jonka avulla Metsähovin radiohäiriöympäristöstä on saatu entistä tarkempaa suuntaavaa informaatiota. Metsähovin radiohäiriöitä ja niiden vaikutuksia on tutkittu hyvin monipuolisesti Jukka-Pekka Porkon diplomityössä ”Radio frequency interference in radio astronomy” [1].

Häiriömonitoroinnissa käytetyn järjestelmän pääpiirteinen kokoonpano on esitetty kuvassa 3.3. Häiriömonitorin antennina toimii rakennuksen katolle itä-siipeen asennettu log-periodinen LPD 500-2000 –antenni (kuva 3.4), jonka vahvistus on noin +9...10 dBi:n luokkaa 400 - 2000 MHz:n taajuusalueella. Antenni sijaitsee noin 20 metrin etäisyydellä observatorion Cassegrain-radioteleskoopista. Antennin suuntaa ohjaa Yaesu G-800 DX ohjausjärjestelmä, jota ohjataan tietokoneella RS-232 –liitännän kautta. Antennin vastaanottama signaali vahvistetaan Mini Circuits ZX60-2531M-5 vahvistimella, jonka vahvistus on 30 dB. Vahvistin on yhdistetty mittauksiin käytettävään Agilent FieldFox N9912A-yhdistelmäanalysaattoriin noin 45 metrin pituisella Suhner Sucoflex RG-214 koaksiaalikaapelilla, jonka vaimennus on noin 0,4 dB/m 2 GHz:n taajuudella. Mittalaitteena käytettävää Agilent Fieldfox –yhdistelmäanalysaattoria (kuva 3.4) ohjataan tietokoneella Ethernet-yhteyden kautta ja laitteella mitatut mittaustulokset tallennetaan tietokoneelle niiden analysointia ja jälkikäsittelyä varten. [1, pp. 32-33] [38]



Kuva 3.3: Metsähovin RFI-monitorointijärjestelmän kokoonpano. [38, p. 8]

Kuva 3.4: Häiriömonitoroinnissa käytettävä LPD 500-2000 antenni sekä itse näytteistykseen käytettävä Agilent FieldFox N9912A yhdistelmäanalysaattori. Kuvat muokattu lähteestä [38]

Monitorointilaitteistolla mitattava taajuuskaista kattaa suurimman osan Metsähovissa käytössä olevien vastaanottimien käyttämistä IF-välitaajuuksista. Monitoroitava 400-2000 MHz:n taajuusalue on jaettu 100 MHz:n osiin (16kpl), joista jokainen pyyhkäistään kymmenen kertaa käyttäen spektrianalysaattorin ”max hold”-asetusta. Mittausdataa kerätään neljästä pääilmansuunnasta kääntäen häiriömonitorin antennia mittausten välissä. Kokonainen mittaus taas koostuu 64 erillisestä koko taajuusalueen kattavasta pyyhkäisystä. Mittauksissa käytetään 3 kHz:n resoluutiokaistanleveyttä ja 1 kHz:n videokaistanleveyttä. Yhdessä 1000 mittauspistettä kattavassa 100 MHz:n pyyhkäisyssä kuluu 6,1 sekuntia eli kokonaiseen 64 pyyhkäisyä kattavaan mittaukseen kuluu noin 1,5

tuntia [42]. Pienimpien järjestelmällä havaittavissa olevien signaalien tehotasot ovat luokkaa -130...-140 dB(W/m^2/Hz). [1]

3.3.1 Agilent FieldFox N9912A

Mittausten näytteistyksen suorittava Agilent FieldFox 9912A on kompakti akkukäyttöinen yhdistelmäanalysaattori, joka sisältää toimintoja kaapelien ja antennien analysointiin sekä esimerkiksi kanavantehon mittauksiin. Häiriömonitoroinnissa laitetta kuitenkin käytetään sen spektrianalysaattorimoodissa. Laitteen häiriömonitoroinnin kannalta tärkeimmät ominaisuudet on listattu taulukossa 3-2. Laite toimii pyyhkäisevänä spektrianalysaattorina eli se pyyhkäisee mitattavan kaistan mitaten tiettyä yksittäistä taajuutta ohuen kaistanpäästösuodattimen läpi (luku 2.6). Laitetta ohjataan tietokoneen välityksellä Ethernet-yhteyden yli käyttäen SCPI-standardin mukaisia komentoja. Laitteesta ja sen etäohjaamisesta on löydettävissä lisää tietoa lähteistä [44] sekä [45].

Agilent FieldFox 9912A			
Mitattavissa oleva taajuusalue	100 kHz - 4 GHz	**RF sisääntulon VSWR**	1,5:1
DANL (Esivahvistimella)	-130 dBm (-148 dBm)	**-60/-3dB valitsevuus**	4:1
Mitattavissa oleva amplitudialue	DANL...+20 dBm	**Sisääntulo-vaimennin**	0 - 31 dB (1 dB:n pykälissä)
Dynaaminen alue	96 dB	**Suurin sisääntuloteho**	+27 dBm, eli 0.5 W ±50 VDC
Amplitudin tarkkuus	±0,5 dB luokkaa normaaliolosuhteissa	**SHI (-30dBm signaalilla)**	< -70 dBc, +40 dBm
Taajuusreferenssin stabiliteetti	±2 ppm ±1 ppm/v ±1 ppm (-10 ... 55 °C)	**TOI (-30dBm signaalilla)**	< -96 dBc, +18 dBm

Taulukko 3-2: Agilent FieldFox 9912A yhdistelmäanalysaattorin tärkeimmät ominaisuudet [45]

3.4 Suojautuminen häiriöitä vastaan

Radioastronomiset mittaukset ovat erittäin herkkiä ulkoisia häiriösignaaleja kohtaan. Metsähovin radio-observatoriossa radioastronomisia mittauksia suoritetaankin vain radioastronomiaan varattujen taajuuskaistojen sisällä. Myös observatorion infrastruktuurilla on pyritty parantamaan mittausten häiriösuojaustasoa. Esimerkiksi observatorion sisäiset signaalitiet on toteutettu tuplasuojatuilla RG-223 ja RG-214 –koaksiaalikaapeleilla niiden häiriöttömyyden varmistamiseksi. Kyseisten kaapelien vaimennus ulkoisia

signaaleja kohtaan on luokkaa 95%. Metsähovista lähtevät ja Metsähoviin tulevat tietoliikenneyhteydet on toteutettu häiriöttömillä optisilla kuituyhteyksillä. [1] [38]

Mittauksia pyritään suojaamaan myös erilaisten menettelytapojen ja sääntöjen avulla. Esimerkiksi selkeitä häiriöitä tuottavien laitteiden, kuten mobiililaitteiden tai loisteputkivalojen käyttö on kielletty observatorion alueella. Mobiililaitteiden käyttöä myös valvotaan Metsähoviin kesällä 2011 asennetulla GSM-monitorointijärjestelmällä. Järjestelmä lähettää varoitusviestin Metsähovin tekniselle työntekijälle, jos mitatut signaalitasot ylittävät ennalta määrätyn raja-arvon [41, p. 14]. Myös mikroaaltouunien käyttö on kielletty, kun observatoriossa tehdään geodeettisiä mittauksia lähellä mikroaaltouunien 2,4 – 2,5 GHz:n toiminta-aluetta. [1] [38]

Häiriösignaaleja myös monitoroidaan aktiivisesti tosin lähinnä suoritettavien mittausten IF-välitaajuudella. Metsähovin nykyinen häiriösuojautuminen perustuukin suurilta osin passiiviseen spektrin monitorointiin ja löydettyjen, häiriöistä vapaiden taajuusalueiden hyödyntämiseen. Häiriöiden mittausta ja tulkintaa edesautetaan ylläpitämällä hyviä kontakteja Suomessa radiotaajuista allokaatiota ylläpitävään auktoriteettiin eli Viestintävirastoon. [38]

Luku 4

4 Ohjelmistoradio (SDR)

Tässä luvussa esitetään ohjelmistoradion (engl. Software-defined radio, SDR) perusperiaatteet lähtien liikkeelle ohjelmistoradion historiasta sekä eri tasoisten ohjelmistoradiojärjestelmien välisistä arkkitehtuurieroista. Luvussa esitetään myös Ettus Research:n nykyinen USRP-laitekanta. Tämän jälkeen luvussa käydään läpi työssä käytettävän Ettus Research N210 USRP:n osakokonaisuuksien, kuten virtalähdekomponenttien, signaalin ylös- ja alasmuuttamisessa käytettävien DUC- ja DDC-piirien, signaalin näytteistyksessä käytettävien ADC- ja DAC –piirien, kellopiirien, laitetta ohjaavan FPGA:n sekä laitteen etupäänä käytettävien tytärkorttien oleelliset piirteet.

Liang Zhou on kuvannut diplomityössään ”USRP2-Based Software Defined Radar Receiver” [46] ohjelmistoradion toimintaa hyvin monipuolisesti lähtien liikkeelle ohjelmistoradion osakokonaisuuksien suorittamien funktioiden, kuten näytteistyksen, suodatuksen sekä alasmuuntamisen matemaattisesta teoriasta. Työ esittelee myös laitteen komponenttien epäideaalisuuksia ja niiden vaikutuksia esimerkiksi laitteen taajuustason tarkkuuteen ja synkronisointiin. Kyseiseen työhön tutustumista suositellaankin edellä mainittujen kohtien osalta tässä työssä esitettyä matemaattisemman ja osiltaan tarkemman tarkastelun mahdollistamiseksi.

4.1 Ohjelmistoradion historiaa

Ohjelmistoradion arkkitehtuurin alkuperäisen konseptin esitti Joseph Mitola vuonna 1992 julkaisussaan ”Software Radio: Survey, Critical Analysis and Future Directions” [47]. Hänen kuvauksensa mukaan ohjelmistoradio (tarkemmin ohjelmistomääritelty radio eli SDR) tarkoittaa radiojärjestelmää, jossa kanavanmodulaation aallonmuodot on määritelty ohjelmallisesti. Konseptin ideana on mahdollistaa laitteen erittäin monipuolinen konfigurointi ja yhden radiojärjestelmän käyttö hyvin erilaisissa käyttökohteissa. Vaikka ohjelmistoradion termi kehitettiinkin vasta 1990-luvulla, on radiojärjestelmien ohjelmallista toteuttamista tutkittu jo 1970-luvulta lähtien lähinnä eri maiden asevoimien sisällä [48]. Ohjelmistoradiojärjestelmät perustuivat 1990-luvulla lähinnä suljettuihin arkkitehtuureihin ja ohjelmistoihin, mikä rajoitti niiden käyttöä tieteellisellä ja yksityisellä sektorilla. Ohjelmistoradiojärjestelmät ovat kuitenkin kehittyneet arkkitehtuurin ja ohjelmistonsa puolesta entistä avoimempaan suuntaan, mikä on mahdollistanut laitteiden käytön ja kehittämisen laajenemisen myös siviilipuolelle. [49, pp. 8-9] [50, p. 9] [51]

Todellinen ohjelmistoradio (Software Radio, SR) on ohjelmistolla määritellystä radiosta (SDR) lähtöisin oleva konsepti, jonka määritelmä on IEEE:n mukaan ”Radio in which some or all of the physical layers functions are software defined” [52, p. 1], eli vapaasti suomennettuna radiojärjestelmä, jossa kaikki tai osa fyysisen tason funktioista on määritelty ohjelmistossa. Ohjelmistoradion ja ohjelmistolla määritellyn radion selkein ero on se, että ohjelmistomääritelty radio (SDR) sisältää erillisen viritettävän radiovastaanottimen, jolla signaalit sekoitetaan alemmalle taajuudelle ja näytteistys suoritetaan vasta IF-välitaajuudella, kun taas todellisessa ohjelmistoradiossa (SR)

näytteistys tapahtuu jo suoraan RF-taajuudella. Käytännössä ohjelmistoradion termiä ja lyhennettä SDR käytetään kuitenkin yleisesti nykyään mistä tahansa ohjelmistoradiojärjestelmästä riippumatta siitä, onko kyseessä todellinen ohjelmistoradio vai ohjelmistolla määritelty radiojärjestelmä. [48, pp. 2-5] [52] [53]

Ohjelmistoradio mahdollistaa laitteiden ominaisuuksien ja toimintojen päivittämisen suoraan ohjelmistopäivitysten avulla. Ohjelmistoradio onkin avainteknologia, jota voidaan käyttää useissa käyttökohteissa koko langattoman kommunikaation alalla. Se mahdollistaa esimerkiksi uusien standardien, taajuusalueiden tai modulaatioiden käyttöönoton ilman tarvetta fyysisen tason muutoksille itse kohteissa käytettävien laitteiden osalta. Ohjelmistoradiojärjestelmien korkeat aloituskustannukset, prosessointitehon vaatimukset sekä korkea tehonkulutus rajoittavat kuitenkin nykyään järjestelmien käyttökohteita. [52] [51] [54]

Ohjelmistoradiojärjestelmiä käytetäänkin nykyään lähinnä langattoman dataliikenteen tukiasemissa ja tutkimuskäytössä tai muissa staattisissa käyttökohteissa, joissa laitteiden päivitettävyydellä ja monipuolisuudella saavutettavat edut ylittävät laitteiden korkeisiin aloituskustannuksiin ja suureen tehonkulutukseen liittyvät heikkoudet. Nopeasti kehittyvät tietoliikenneyhteydet ja palvelut ovat nostaneet operaattorien tukiasemien päivityksiin liittyviä kuluja huomattavasti, mikä on tehnyt ohjelmistoradiojärjestelmistä entistä käytännöllisempiä vanhojen rautapohjaisten järjestelmien korvaamisessa. Staattinen taajuusallokaatio rajoittaa kuitenkin nykyään ohjelmistoradiojärjestelmien käyttöönottoa kaupallisissa kohteissa [55, p. 8]. Käytetyt standardit ja taajuusallokaatio tulevatkin todennäköisesti muuttumaan entistä dynaamisempaan suuntaan tulevaisuudessa, mikä onkin ensisijaisen tärkeää älykkäiden radiojärjestelmien, kuten kognitiivisen radion yleistymisen kannalta. [52] [51]

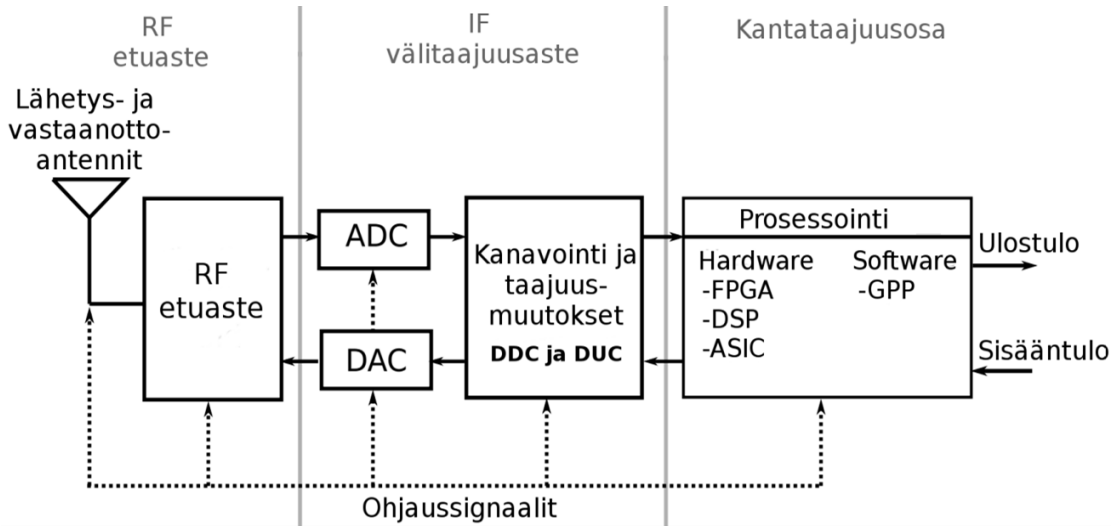
4.2 Ohjelmistoradion toiminta

Ohjelmistoradion perusperiaate on toteuttaa mahdollisimman suuri osa aiemmin analogisilla komponenteilla, kuten vahvistimilla, suodattimilla, sekoittajilla, modulaattoreilla sekä detektoreilla toteutetuista osista rekonfiguroitavan digitaalielektroniikan, kuten yleiskäyttöisten prosessorien (engl. General Purpose Prosessor, GPP) tai DSP-, FPGA- ja ASIC –piirien sekä niiden toimintaa ohjaavan ohjelmiston avulla. Yleensä ohjelmistoradiojärjestelmät toimivat niin, että itse ohjelmistoradio sisältää RF-etuastekomponentit, näytteistykseen tarvittavat AD/DA-muuntimet sekä näitä ohjaavan ja laitteen kommunikaatiosta huolehtivan FPGA-piirin. Koko järjestelmää ohjataan useimmiten ulkoisesti tietokoneella jonkin yleiskäyttöisen dataliitännän, kuten USB:n tai Ethernet:n välityksellä. Monet ohjelmistoradiojärjestelmät, kuten tässä työssä käytetty Ettus Research N210, toimivat oletusajureillaan niin, että kaikki signaalinkäsittely suoritetaan vasta järjestelmää ohjaavalla tietokoneella. Tällöin itse ohjelmistoradio toimii siis vain ulkoisesti ohjattavana mittajärjestelmän etupäänä. [56, pp. 10-11]

4.3 Ohjelmistoradion arkkitehtuuri

Ideaalinen ohjelmistoradiojärjestelmä sisältäisi vastaanotinantennin lisäksi pelkästään signaalin näytteistyksen suorittavat (ideaaliset) komponentit ja kaikki signaalille tehtävät operaatiot suoritettaisiin ohjelmallisesti. Ideaalinen ohjelmistoradio kykenisi siis moduloimaan ja demoduloimaan signaalia millä tahansa taajuudella ja taajuuskaistal-

la. Nykyisin kaupallisesti saatavilla olevien A/D-muuntimien epäideaalisuudet, kuten näytteistyksen hitaus, dynaamisen alueen rajallisuus sekä korkea energiankulutus, rajoittavat toteutettavissa olevien ”ideaalisten” ohjelmistoradiojärjestelmien avulla käytettävissä olevaa kaistanleveyttä. Ohjelmistoradiojärjestelmissä käytetäänkin edellä mainittujen heikkouksien kompensointiin yleisesti radiotekniikassa käytettäviä analogisia RF-etuastekomponentteja. Tällaisen realistisesti toteutettavissa olevan ohjelmistoradiojärjestelmän kokoonpano on esitetty kuvassa 4.1. [56, pp. 6-9] [53, pp. 5-8]



Kuva 4.1: Realistisesti toteutettavissa olevan ohjelmistoradiojärjestelmän arkkitehtuuri. Kuva muokattu lähteistä [54, p. 6] ja [57, p. 12]

Ohjelmistoradion kuvassa 4.1 esitetty arkkitehtuuri voidaan jakaa sen osien sisäisen toimintataajuuden mukaisesti kolmeen osioon: radiotaajuiseen RF-etuasteosaan, välitaajuudella toimivaan IF-välitaajuusosaan sekä signaaliprosessorin taajuudella toimivaan kantataajuusosaan. Etuasteosa sisältää vastaanotto- ja lähetysantennit sekä lähinnä analogisia, useimmiten ohjelmallisesti ohjattavia etuastekomponentteja, kuten vahvistimia, kytkimiä ja vaimentimia. Lisäksi etuasteosa sisältää sekoittajia, joiden avulla vastaanotettujen signaalien taajuus muunnetaan IF-välitaajuudelle niiden näytteistystä varten. [53, pp. 6-7]

Etuasteosaa seuraava välitaajuusosa sisältää analogia-digitaali-muuntimen (engl. Analog to Digital Converter, ADC), jolla välitaajuudelle muunnettu signaali näytteistetään, sekä digitaalisen alasmuuntimen (engl. Digital Down Converter, DDC), jonka avulla näytteistetty signaali alasmuutetaan välitaajuudelta kantataajuudelle sen prosessointia varten. Lähetyspuolella välitaajuusosa toteuttaa vastaavat toiminnot käänteisesti digitaalisen ylösmuuntimen (engl. Digital Up Converter, DUC) ja digitaali-analogia-muuntimen (engl. Digital to Analog Converter, DAC) avulla. Välitaajuusosa toimii siis laitteen analogisen ja digitaalisen puolen rajapintana. [54, p. 7]

Välitaajuusosaa seuraa digitaalielektroniikasta koostuva kantataajuusosa, joka sisältää FPGA-, DSP- tai ASIC-piireistä sekä mahdollisista mikrokontrollereista tai mikroprosessoreista koostuvan prosessointiyksikön. Kaikki ohjelmistoradion sisällä raudassa toteutettu signaalinkäsittely suoritetaan kantataajuusosassa. Yleensä kantataajuusosa sisältää myös liitäntärajapinnat, jotka mahdollistavat laitteen kommunikoinnin esimerkiksi ohjelmistoradion toimintaa ohjaavan tietokoneen kanssa. [54, p. 7]

4.3.1 Ohjelmistoradiotyyppien määrittely tasojen mukaan

Ohjelmistoradioihin keskittyvällä ”The Wireless Innovation Forum”-nimisellä sivustolla on esitetty jaottelu, jossa ohjelmistoradiotyypit on jaettu niiden ominaisuuksien mukaisiin tasoihin seuraavalla tavalla: [52] [48, pp. 4-5]

Taso 0: HW-radio (Hardware Radio, HR)
Perinteinen radiojärjestelmä, joka on toteutettu pelkästään elektronisilla komponenteilla. Laitteen toimintaparametreja ei ole mahdollista muuttaa tekemättä fyysisiä muutoksia itse laitteeseen.

Taso 1: Ohjelmistolla ohjattu radio (Software Controlled Radio, SCR)
Radiojärjestelmän ohjausfunktiot on toteutettu ohjelmallisesti, mikä mahdollistaa joidenkin sen yksittäisten parametrien, kuten tehotasojen, muokkaamisen. Järjestelmä ei kuitenkaan mahdollista käytettävän taajuusalueen tai modulaation tyypin muuttamista ohjelmallisesti.

Taso 2: Ohjelmistolla määritelty radio (Software Defined Radio, SDR)
Ohjelmistolla määritelty radiojärjestelmä on monimutkainen, mutta realistisesti toteutettavissa oleva radiojärjestelmä. Yleisesti ohjelmistoradiolla tarkoitetaankin nykyään useimmiten juuri tämän tyyppistä radiojärjestelmää. Ohjelmistolla määritelty radiojärjestelmä mahdollistaa useiden laitteen parametrien, kuten taajuusalueen, modulaation sekä käytetyn kaistanleveyden ja standardien muuttamisen täysin ohjelmallisesti. Myös järjestelmän toiminnan laajentaminen on siis mahdollista toteuttaa ohjelmistopäivitysten avulla. Järjestelmä tarvitsee kuitenkin erillisen RF-etuasteen. Järjestelmän toimintaa rajoittavatkin yleensä sen etuasteen komponentit, A/D- ja D/A-muuntimet sekä edellä mainittujen rajallinen kaistanleveys ja dynaaminen alue. Käytettävän taajuusalueen muuttaminen voikin edellyttää RF-etuasteen sisältämien osien fyysistä vaihtamista. Ohjelmistolla määritellyt radiojärjestelmät ovatkin useimmiten rakenteeltaan modulaarisia edellä kuvattujen muutosten yksinkertaistamiseksi.

Taso 3: Ideaalinen ohjelmistoradio (Ideal Software Radio, ISR)
Ideaalisen ohjelmistoradion ja SDR:n suurin ero on se, ettei ideaalinen ohjelmistoradio sisällä erillistä RF-etuastetta. Antennilta tuleva signaali näytteistetään suoraan RF-taajuudella ja aiemmin RF-etuasteen toteuttamat funktiot, kuten taajuusalueen muutokset toteutetaan ohjelmallisesti. Ideaalisen ohjelmistoradion laitteistokokoonpano koostuu pelkästään antenneista, AD/DA–muuntimista sekä digitaalisista signaaliprosessointikomponenteista. Ideaalisen ohjelmistoradion toteuttaminen on ongelmallista lähinnä kustannus- ja tehonkulutussyistä, mistä johtuen järjestelmät toteutetaankin pääosin ohjelmistomääriteltyinä. Ideaalisen ohjelmistoradion määritelmä on hyvin lähellä Joseph Mitolan alkuperäistä ”todellisen ohjelmistoradion” määritelmää.

Taso 4: Lopullinen ohjelmistoradio (Ultimate Software Radio, USR)
Lopullisen ohjelmistoradion määritelmä ei huomioi teknologisia rajoitteita, vaan se on idealistinen kuvaus radiojärjestelmästä, jossa kaikki toiminnallisuus on toteutettu ohjelmallisesti. Lopullinen ohjelmistoradio ei esimerkiksi tarvitse erillistä antennia, eikä sen käytössä olevalla taajuusalueella ole rajoituksia. Lopullinen ohjelmistoradio onkin käytännössä mahdoton toteuttaa.

Kognitiivinen radio (Cognitive Radio, CR)
Kognitiivinen radio on ITU:n määritelmän mukaan: “A radio or system that senses, and is aware of, its operational environment and can dynamically and autonomously adjust its radio operating parameters accordingly.“ [58, p. 14]. Kognitiivinen radio tarkoittaa siis radiojärjestelmää, joka on tietoinen ympäristöstään ja kyvyistään, sekä kykenee muuttaamaan fyysisen tason parametrejään, toimintaansa ja toimintataajuuttaan parantaakseen yhteensopivuuttaan kyseisessä radioympäristössä. Lisäksi järjestelmä kykenee oppimaan, mikä mahdollistaa sen yhteensopivuuden ylläpitämiseen muuttuvassa radioympäristössä. Oppiminen voi perustua esimerkiksi järjestelmän itse suorittamiin mittauksiin ja/tai ulkoisesti muilta kognitiivisilta järjestelmiltä vastaanotettuihin tietoihin. Kognitiivisten radiojärjestelmien tärkeimmät sovelluskohteet ovat spektrinkäytön sekä järjestelmien keskinäisen yhteensopivuuden tehostaminen. Spektrinkäytön tehostaminen perustuu lähinnä käyttäjistä tyhjien spektrialueiden havaitsemiseen ja hyödyntämiseen [59]. Järjestelmien välistä yhteensopivuutta kognitiivinen radiojärjestelmä taas voi tehostaa esimerkiksi toimimalla välityspalvelimena eri taajuusalueilla tai modulaatioilla toimivien laitteiden välillä. [60] [61, pp. 2-9]

4.4 USRP (Universal Software Radio Peripheral)

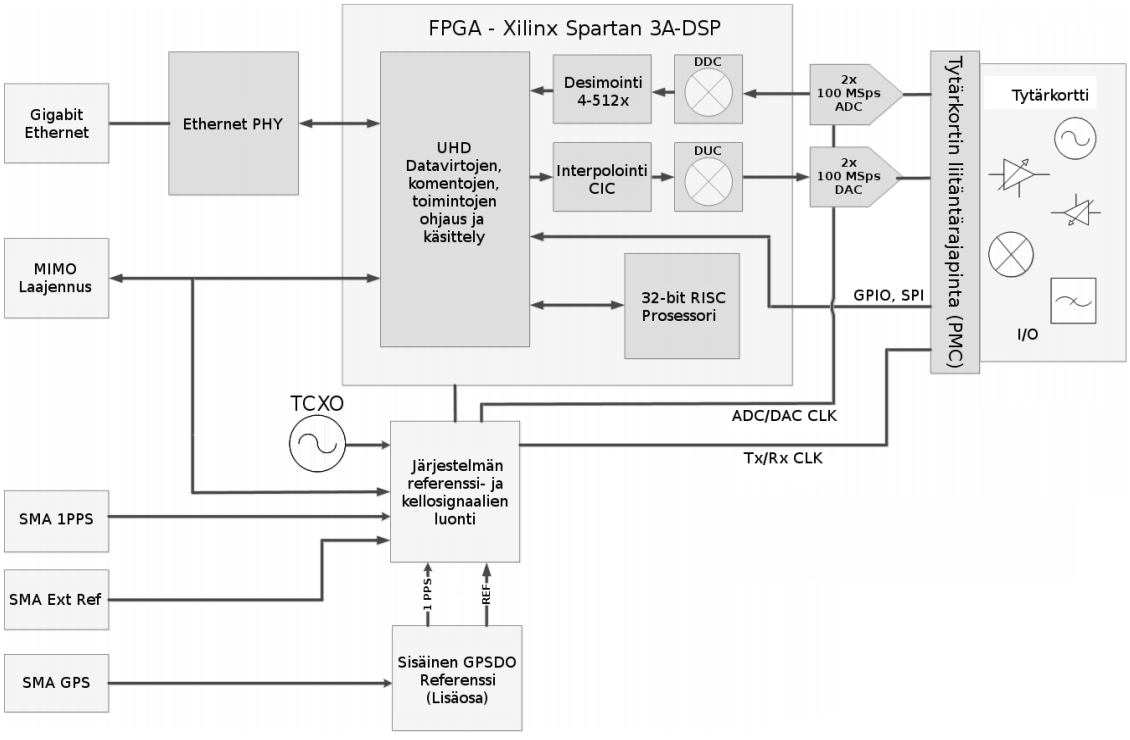
The Universal Software Radio Peripheral eli USRP on tietokonepohjainen, avoimeen suunnitteluun perustuva ohjelmistoradiokehitysalusta, jota kehittävät Ettus Research LLC sekä yhtiön nykyinen omistaja National Instruments. USRP-tuoteperhe on kohdistettu tutkimuskäyttöön yksityisille laboratorioille, yliopistoille sekä yksityishenkilöille. Tuoteperheen kehityksestä vastaa Matt Ettus. USRP-laitteita ohjataan tietokoneen välityksellä avoimen lähdekoodin UHD-ajurilla ja järjestelmän signaalinkäsittely toteutetaan useimmiten GNU Radio –ohjelmistoon perustuvilla tietokoneohjelmilla. USRP ”Embedded”-sarjan kehitysalustat tosin sisältävät myös erillisen ARM-mikroprosessorin, joka mahdollistaa laitteen itsenäisen käyttämisen ilman tarvetta erilliselle, järjestelmää ohjaavalle tietokoneelle. USRP-tuotteet on suunniteltu avoimen lähdekoodin periaatteella ja niihin liittyvät kytkentäkaaviot, komponenttien sijoittelu sekä ajurit ovat vapaasti laitteiden käyttäjien saatavilla [62]. Järjestelmän avoimuus yksinkertaistaa esimerkiksi laitteen etuasteena käytettävien tytärkorttien itsenäistä jatkokehittämistä. [46, p. 32] [63]

USRP-laitteiden ytimenä toimii emolevy, joka sisältää laitteen keskeisimmät toiminnot toteuttavan FPGA-piirin sekä signaalin näytteistykseen tarvittavat A/D- ja D/A-muuntimet. Emolevy sisältää useimmissa USRP-laitteissa myös etuasteen rajapintana toimivat PMC-liittimet, joihin voidaan asentaa haluttua taajuusaluetta ja toiminnallisuutta vastaava, laitteen etuasteena toimiva, tytärkortti. B1x0-, B2x0- ja QR210-sarjan USRP-laitteissa etuaste on tosin integroituna suoraan laitteen emolevylle. Tytärkortista riippuen USRP-laitteiden vastaanotto- ja lähetystaajuudet voivat olla välillä DC - 6 GHz. USRP-laitteet ovat usein MIMO-yhteensopivia, mikä mahdollistaa useamman laitteen käyttämisen rinnakkain, parantaen järjestelmän ominaisuuksia monissa käyttökohteissa. Lisäksi useimmat USPR-laitteet sisältävät liitännän ulkoisen referenssikellosignaalin tai GPS-moduulin liittämiseksi laitteeseen käyttökohteissa, joissa laitteen sisäisen kelloreferenssin tarkkuus ei riitä. Yhteys tietokoneeseen muodostetaan USRP-mallista riippuen USB- tai Ethernet-väylän, tai erillisen tietokoneeseen asennettavan PCIe-kortin ja liitäntäkaapelin avulla. Liitteessä 1 on esitetty taulukko, jossa Ettus Research:n nykyisiä USRP-laitteita on vertailtu niiden edellä mainittujen ominaisuuksien osalta. [63] [64, pp. 23-29]

USRP-mallien välillä on huomattavia eroja niiden liitäntärajapintojen ja näytteistyksen komponenttien lisäksi myös niiden sisältämien FPGA–piirien tehokkuuden sekä vapaiden resurssien määrän suhteen. Näitä eroja on havainnollistettu valittujen USRP-laitteiden välillä liitteessä 2 esitetyssä taulukossa. USRP N210:n Xilinx XC3D3400A FPGA on suorituskyvyltään kohtuullisen tehokas piiri ja se mahdollistaa varsin monipuolisten signaalinkäsittelyfunktioiden rautapohjaisen suorittamisen. Uudemmat B2x0- ja X3x0 –sarjojen ohjelmistoradiojärjestelmät ovat tosin vielä huomattavasti kehittyneempiä logiikkaelementtiensä, RAM-muistinsa sekä signaalinkäsittelyssä tarpeellisten DSP-moduuliensa lukumäärän suhteen. N200- ja B200- sarjan USRP-laitteiden sisältämä FPGA on mahdollista ohjelmoida Xilinx:n ilmaisten ISE WebPACK-ohjelmointityökalujen avulla. USRP N210-, B210- ja X3x0- sarjan USRP-laitteiden FPGA-piirien ohjelmointi sen sijaan on mahdollista vain Xilinx:n kehitystyökalujen maksullisella versiolla. [63] [65, p. 15]

4.5 Ettus Research USRP N210

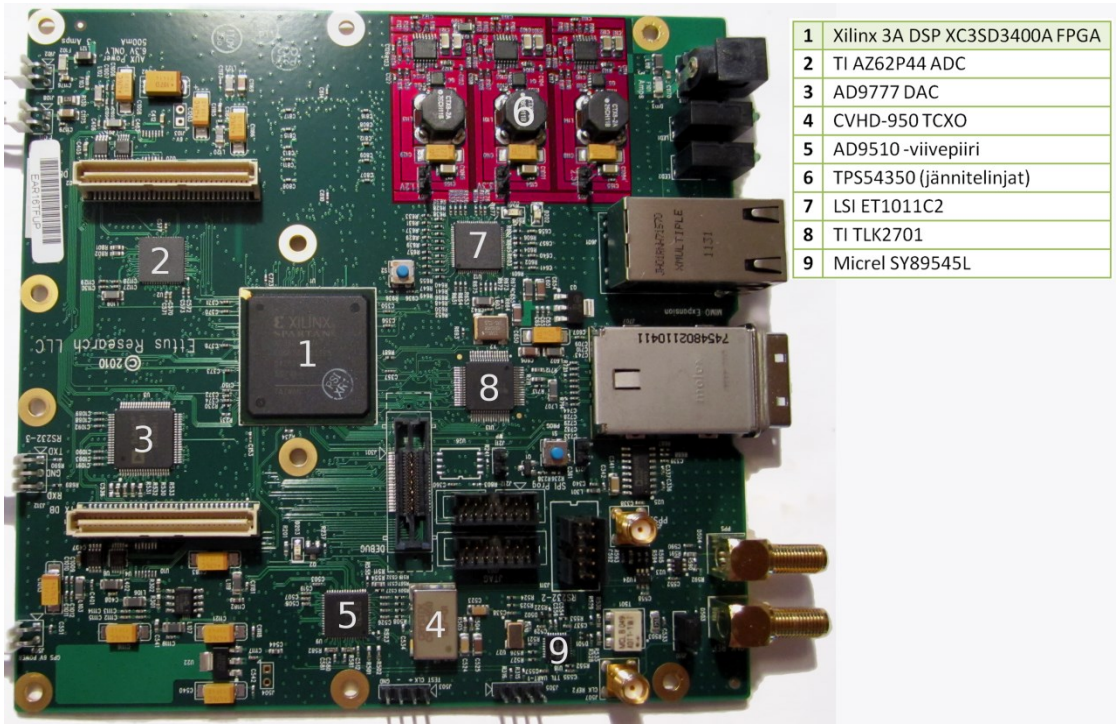
Työssä käytettävä USRP N210 on Ettus Research:n vuonna 2010 julkaisema ohjelmistoradiokehitysalusta, jonka keskeisimmät päivitykset aiempiin USRP-malleihin nähden ovat sen reilusti suurempi FPGA-kapasiteetti, kehittyneemmät näytteistyksen komponentit, emolevylle integroitu kohtuullisen tarkka TCXO-kelloreferenssi sekä laitteen kehittyneemmät liitännät. USRP N210–ohjelmistoradiojärjestelmän sisäistä toimintaa ja signaalien etenemistä selventävä kaavio on esitetty kuvassa 4.2. [63]



Kuva 4.2: USRP N210 sisäisten signaalien eteneminen. Kuva muokattu lähteestä [66]

4.5.1 Emolevy

Ettus Research USRP N210 emolevy on esitetty kuvassa 4.3. Sen keskeisimmät osat ovat kuvan keskellä näkyvä Xilinx Spartan 3A DSP - XC3SD3400A FPGA –piiri, FPGA:n vasemmalla puolella oleva Texas Instruments AZ62P44 ADC –piiri ja sen alapuolella oleva Analog Devices AD9777 DAC –piiri, emolevyn alareunassa hieman FPGA:n oikealla puolella sijaitseva Crystek:n CVHD-950 TCXO-kelloreferenssi sekä tämän vasemmalla puolella sijaitseva kellosignaalien jakamisessa käytetty AD9510-viivepiiri. Lisäksi emolevyllä erottuvat selvästi laitteen yläreunassa näkyvät laitteen jännitelinjoja reguloivat hakkuriregulaattorit, näiden alapuolella näkyvä LSI:n ET10-11C2 –Ethernet-ohjain sekä tämän alapuolella näkyvä, laitteen dataliikenteestä huolehtiva, Texas Instruments TLK2701 –lähetinvastaanotinpiiri. [62]



Kuva 4.3: USRP N210 emolevy

4.5.2 FPGA

FPGA (engl. Field-Programmable Gate Array) on digitaalinen mikropiiri, jonka sisältämien komponenttien väliset kytkennät on toteutettu ohjelmoitavilla logiikkaporteilla. Ohjelmoitavien logiikkaporttien (engl. Configurable Logic Block, CLB) lisäksi FPGA-piirit sisältävät useimmiten RAM-muistia, erilaisia logiikkaelementtejä sekä nykyään usein myös monimutkaisempia kokonaisuuksia, kuten ADC-, DAC- ja DSP-piirejä sekä mikroprosessoriytimiä. FPGA-piirien käyttö on yleistynyt huomattavasti viimeisen vuosikymmenen aikana piirien nopeuden ja loogisen kapasiteetin sekä edellä kuvatun integraation kehittymisen ansiosta. [67] [68]

FPGA-piirien toiminta eroaa normaaleista yleiskäyttöistä prosessoreista selvästi, sillä yleiskäyttöiset prosessorit (GPP) suorittavat ohjelmakoodin rivi ja käsky kerrallaan sarjassa, kun taas FPGA–piirit käsittelevät kaikki logiikkaelementtiensä operaatiot rinnakkain, mikä tehostaa rinnakkaistuvien algoritmien suoritusta. Ohjelman rauta-

pohjainen toiminta tosin rajoittaa suoritettavissa olevan ohjelmakoodin kokoa, sillä ohjelmoitavia logiikkasoluja on FPGA–piirissä aina ennalta rajattu määrä. Solujen määrää ylittävää koodia ei siis pystytä edes lataamaan piirille ja toisaalta tyhjäksi jääneitä logiikkasoluja ei voida hyödyntää ohjelman suorituksen aikana. [67] [68]

FPGA–piirejä käytetään niiden tarjoaman tehokkaan rinnakkaisen signaalinkäsittelyn ja monipuolisen muunnettavuuden ansiosta. FPGA-piirien uudelleenkonfiguroitavien logiikkaelementtien avulla on mahdollista tuottaa jopa geneeristä prosessoria vastaava toiminnallisuus [13, pp. 10-15]. FPGA–piirien tehokas ja monipuolinen käyttö edellyttää kuitenkin niiden ohjelmointia vaikeaselkoisella laitteistokuvauskielellä (engl. Hardware Description Language, HDL). FPGA:n ajurina toimiva laitteistokuvauskielisestä koodista koostettu binäärimuotoinen ohjelma tallennetaan piirin ulkopuolella sijaitsevalle muistipiirille, josta se ladataan käynnistysvaiheessa FPGA:n sisältämiin CMOS-konfigurointivipuihin, joiden muodostamien kytkentöjen avulla ohjataan piirin kaikkia funktionaalisia elementtejä ja näiden keskinäistä linkitystä. [67] [68]

Laitteistokuvauskielet eivät ole yhtä abstrakteja kuin perinteiset ohjelmistokielet, joihin radiojärjestelmiä kehittävät suunnittelijat ovat tottuneet. Lisäksi laitteistokuvauskielellä toteutetun koodin syntetisointi ja kääntäminen on useita kertaluokkia hitaampaa verrattuna ohjelmistototeutukseen [68, p. 53]. Yksittäisessä FPGA-ajurin syntetisoinnissa voikin kestää jopa yli puoli tuntia. Syntetisointiin kuluvan ajan minimointi mahdollistetaankin useimmiten tuotettujen FPGA-ajurien simuloinnin ja erillisten testiohjelmien avulla [69]. Algoritmien toteuttaminen FPGA-piirille hidastaa useimmiten koko laitteen kehitystä, mistä johtuen FPGA-piirien valmistajat ovatkin alkaneet kehittää entistä abstraktimpien syntaksien käytön mahdollistavia, System-C ja C/C++ -ohjelmointikielillä toimivia suunnittelutyökaluja. [67] [68]

4.5.3 Kellosignaalit

USRP N210:n sisäiset kellosignaalit luodaan Crystek CVHD-950-100M TCXO–kelloreferenssin avulla käyttäen signaalien jakamiseen vaihelukittuna toimivaa AD9510-viivepiiriä. Viivepiiriltä lähtevä kellosignaali synkronisoidaan 10 MHz:n referenssikelloon, joka saadaan laitteen sisäisen oskillaattorin, erillisen GPSDO-moduulin tai ulkoisen referenssisisääntulon avulla (kuva 4.2). Referenssikello valitaan automaattisesti Micrel SY89545L-multiplekseripiirillä. USRP N210:n sisäisen 10 MHz:n oskillaattorin avulla saavutettava taajuuden tarkkuus on luokkaa 2.5 ppm ja GPSDO-moduulin avulla tätä tarkkuutta voidaan parantaa aina 0.01 ppm:n asti. Vieläkin parempaan taajuuden tarkkuuteen voidaan päästä käyttämällä ulkoista kellosignaalia. Ulkoisen kelloreferenssin taajuus tulee olla joko 5 tai 10 MHz ja sen signaalitason tulee olla välillä 0 – 15 dBm. Signaali jaetaan AD9510–viivepiiriltä eteenpäin laitteen eri osille, kuten FPGA-, ADC- ja DAC –piireille. Laitteessa on myös 1PPS-sisääntulo, jota voidaan käyttää laitteen toimintojen ajoituksen tarkentamiseen sekä useammista ohjelmistoradioista koostuvien järjestelmien keskinäiseen synkronisointiin. [70] [71]

4.5.4 Digitaaliset ylös- ja alasmuuntimet (DUC & DDC)

USRP N210:n sisäinen 100 MHz:n kellosignaali ei ole säädettävissä ohjelmallisesti, mistä johtuen laitteen sisäiset piirit toimivat aina 100 MHz:n taajuudella. Esimerkiksi muiden näytteenottotaajuuksien käyttäminen mahdollistetaan desimoimalla näytteistettyä signaalia laitteen FPGA:n sisältämien digitaalisten alasmuuntajien (engl. Digital

Down-Converter, DDC) avulla. Kompleksinen 14-bittinen näytteistäminen 100 MHz näytteenottotaajuudella tuottaa 2.8 Gbps datavirran, mikä ylittää laitteen Gigabitin Ethernet-liitännän maksiminopeuden. Tästä johtuen pienin valittavissa oleva desimointikerroin on rajoitettu arvoon neljä, mikä rajaa signaalinkäsittelyssä käytettävissä olevan maksiminäytteenottotaajuuden tasoon 25 MHz, joskin 8-bitin näytteistyksellä on mahdollista käyttää myös 50 MHz:n näytteenottotaajuutta. Desimointikertoimen arvo on valittavissa väliltä 4 – 512 ja se valitaan oletuslaiteajureita käytettäessä automaattisesti käyttäjän valitseman näytteenottotaajuuden mukaan. Huomion arvoista on kuitenkin se, ettei näytteenottotaajuutta voida todellisuudessa valita kuin desimointikertoimen rajoittamissa pykälissä. Laiteajureita muuttamalla on tosin mahdollista käyttää edellä mainittua suurempiakin desimointikertoimia tai esimerkiksi poistaa desimointi käytöstä kokonaan. [46, pp. 17-21] [71] [72]

USRP N210:n FPGA sisältää myös kaksi ylösmuunninta (engl. Digital Up-Converter, DUC), joiden tehtävänä on toteuttaa alasmuuntimien käänteinen funktio, eli muuntaa signaali lähetyspuolella kantataajuudelta IF-välitaajuudelle. Digitaaliset ylösmuuntimet toimivat seuraavalla tavalla: sisääntulevan pienellä näytteenottotaajuudella näytteistetyn signaalin lukuarvojen väleihin lisätään nollia, minkä jälkeen signaali suodatetaan käyttämällä FIR–suodattimia. Näin lisätyt nollat saadaan interpoloitua vastaamaan signaalien oletettuja väliarvoja. [71] [72]

4.5.5 Näytteenotto ja syntetisointi (ADC & DAC)

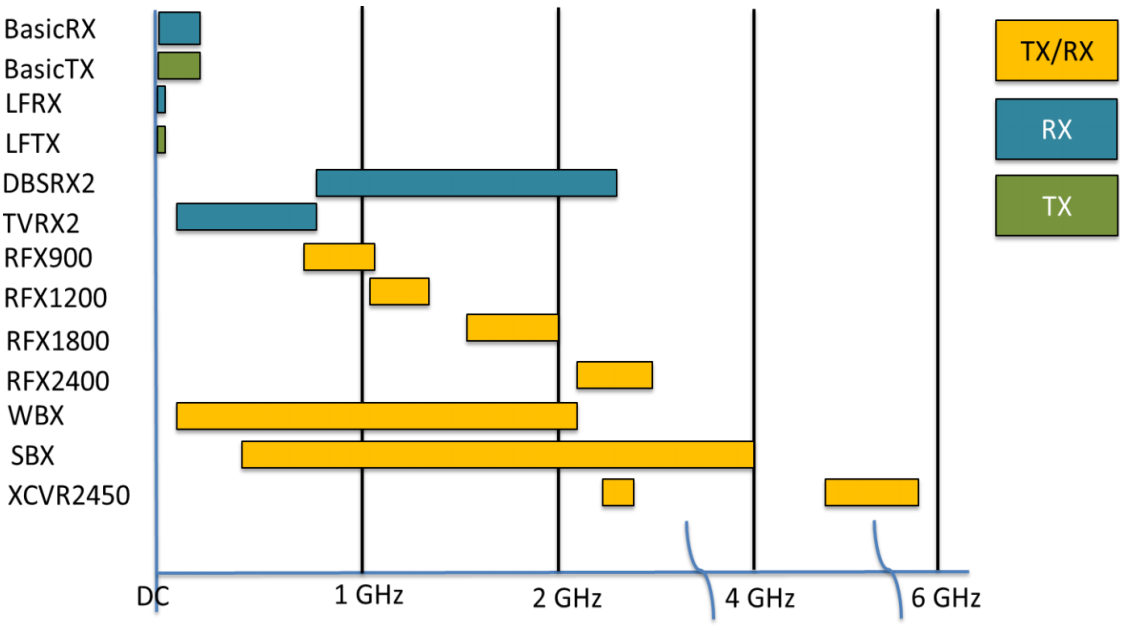
Analogia-digitaalimuunnin (engl. Analog to Digital Converter, ADC) on laite, joka näytteistää ja kvantisoi analogisen sisääntulosignaalin digitaaliseksi bittijonoksi. Saavutettavien kvantisointitasojen määrä riippuu käytettävän ADC:n tyypistä [55, pp. 4-5]. USRP N210:n näytteenotossa käytettävä ADS62P44 -piiri sisältää kaksi 14-bittistä ADC:tä, joiden näytteenottotaajuus on 100 MSps. USRP N210:n sisääntulojen impedanssi on 50 ohmia ja sen sisääntulon huippuarvoinen maksimitaso on 2 V. ADC:n erottelukyvyksi saadaan siis 14-bitillä ja kahden voltin huippuarvoisilla signaaleilla noin 244 µV. [46, pp. 13-16] [71]

Digitaali-analogiamuunnin (engl. Digital to Analog Converter, DAC) on ADC:n käänteisen funktion suorittava laite, eli se mahdollistaa digitaalisen signaali muuntamisen analogiseksi signaaliksi. USRP N210:ssä on kaksi 16-bit:n DAC:ia, jotka toimivat 400 MSps näytteistystaajuudella ja niiden avulla voidaan tuottaa laitteen 50 ohmin differentiaaliseen ulostuloon huippuarvoltaan korkeintaan yhden voltin signaaleja. Lähetyspuolen 400 MSps näytteistystaajuus luodaan interpoloimalla FPGA:lta vastaanotettu 100 MHz:n signaali kertoimella neljä. Digitaali-analogia-muunnoksen jälkeen ulostulosignaaleja voidaan vielä vahvistaa tytärkorttien sisältämien ohjelmallisesti säädettävien lähetysvahvistimien avulla (kuva 4.4). [46, pp. 13-16] [73]

4.5.6 Tytärkortit

USRP:n tytärkortit ovat laitteen RF-etupään toteuttavia lisäosia, jotka määrittävät suoraan, millä taajuuksilla ja taajuuskaistoilla signaaleja pystytään vastaanottamaan tai lähettämään. Tytärkortit sisältävät EEPROM–piirin, jonka avulla ne voidaan tunnistaa automaattisesti, mikä mahdollistaa esimerkiksi korttikohtaisten kalibrointiarvojen automaattisen lataamisen. USRP N210 sisältää kaksi paikkaa tytärkorteille. Pelkät vastaanotin- tai lähetinkortit vievät yhden korttipaikan, kun taas lähetinvastaanotin-

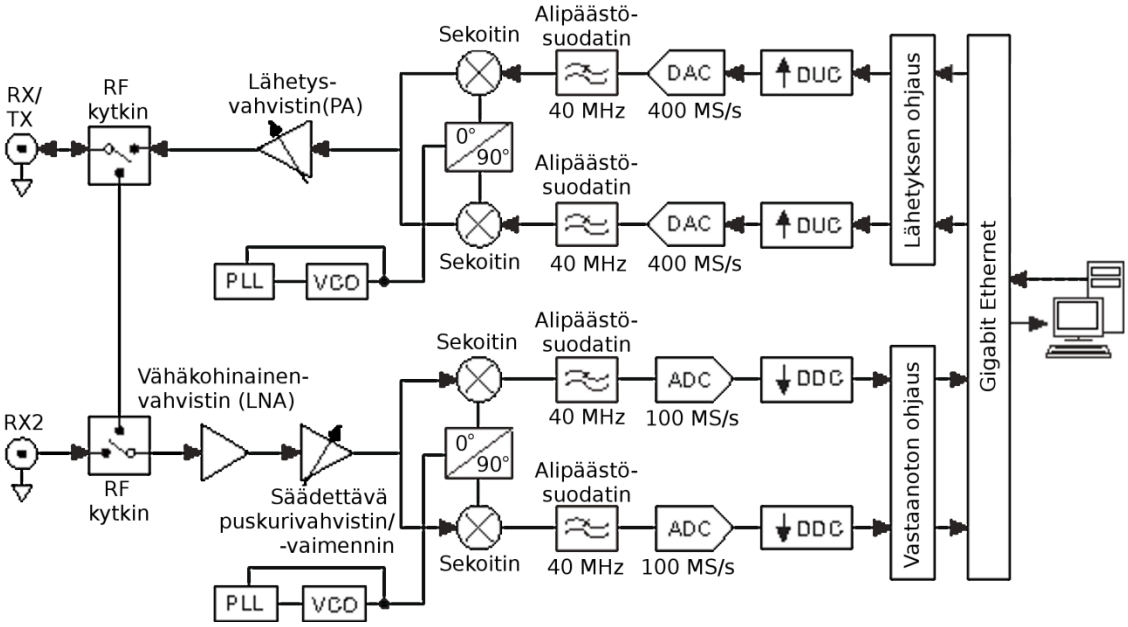
tytärkortit (engl. tranceiver) vievät kaksi korttipaikkaa. Taulukossa 4-1 on esitetty Ettus Research:n valikoimasta löytyvien tytärkorttien taajuusalueiden kattavuus. Tytärkorttien avulla on siis tuettuna lähes koko taajuusalue välillä DC – 6 GHz. [70]



Taulukko 4-1: RF tytärkorttien taajuusalueiden kattavuus [70]

USRP:n avoin konsepti yksinkertaistaa uusien tytärkorttien valmistamista. Järjestelmän avoimuuden ansiosta tytärkortteja onkin mahdollista ostaa myös Ettus Research:n ulkopuolisilta valmistajilta. Esimerkiksi Epiq Solutions myy Bitshark USRP RX (BURX) –tytärkorttia, joka toimii 300 MHz – 4 GHz taajuusalueella. Kortti on varustettu 50 MHz:n ohjelmoitavalla kanavasuodattimella ja se sisältää oman sisäänrakennetun korkean stabiliteetin TXCO-kellopiirinsä. [70] [74]

USRP N210:n ja SBX- tai WBX- tytärkortin muodostaman kokonaisuuden näytteistyksen toimintaa on selvennetty kuvassa 4.4. Kuvasta on oleellista huomata, että lähetys- ja vastaanottokaistat tuotetaan kompleksisina (I/Q), eli yhden kanavan tuottamiseen käytetään kumpaakin ADC:n tai DAC:n kanavaa. Kyseisillä tytärkorteilla ei siis ole mahdollista näytteistää kuin yhtä kanavaa kerrallaan. Toisaalta menetelmä mahdollistaa teoriassa koko 100 MHz:n kaistan näytteistämisen. Kyseisten tytärkorttien tapauksessa käytettävissä oleva näytteistyskaista on kuitenkin rajoittunut korkeintaan 40 MHz:n kaistanleveyteen signaalitien aliasointia vähentävien alipäästösuodattimien ansiosta.



Kuva 4.4: Ettus Research N210 etuasteen toiminta, kun laitteeseen liitetty SBX- tai WBX-tyyppinen tytärkortti. Kuva muokattu lähteestä [75]

4.2.7 MIMO

USRP N210 sisältämä MIMO-liitin mahdollistaa kahden ohjelmistoradiolaitteen ohjaamisen yhden Ethernet-liitännän kautta. MIMO-liitännän kautta kytketty laite näkyy tällöin käyttäjälle samaan tapaan kuin se olisi kytketty omalla, erillisellä Ethernet-yhteydellään. MIMO-liitännän suurin rajoitus on jo muutenkin N210–ohjelmistoradiolaitteiden datayhteyden kaistanleveyttä rajoittava Gigabit:n Ethernet- liitäntä, joka rajaa koko järjestelmällä saavutettavan näytteistysnopeuden yhteensä 25 MSps tasolle 16-bitin näytteistyksellä. [76]

MIMO-yhteyden kautta USRP-laitteet voivat siirtää ja synkronoida keskenään I/Q-dataa sekä ohjaus- ja kellosignaaleja. MIMO-yhteyttä ei kuitenkaan ole rajoitettu pelkästään edellä mainituille signaaleille, vaan sen tarkoituksena on tarjota nopea, minkä tahansa FPGA-signaalin laitteen ulkopuolelle siirtämisen mahdollistava liitäntä. MIMO-yhteyden fyysinen liitäntärajapinta on SF-8088 -tyyppinen miniSAS–portti [77, p. 21]. MIMO-järjestelmä voidaan toteuttaa myös ilman MIMO-liitintä, kun erilliset USRP:t yhdistetään samaan ulkoiseen kelloreferenssiin. Järjestelmä tarvitsee lisäksi 1PPS-signaalin laitteiden toimintojen, kuten näytteistyksen aloituksen samanaikaistamiseksi. Liang Zhou on diplomityössään ”USRP2-based Software Defined Radar Receiver” mitannut edellä kuvatulla järjestelyllä USRP2-ohjelmistoradiolaitteiden välisen saavutettavan synkronisaation virheen olevan alle ±1,0 ns:n luokkaa. Kyseinen USRP2-ohjelmistoradio vastaa sisäiseltä rakenteeltaan hyvin pitkälle tässä työssä käytettyä N210-ohjelmistoradiolaitetta. [46, pp. 23-25]

Luku 5

5 Häiriömonitoroinnin mittausohjelmisto

Tässä luvussa esitellään työssä toteutetun häiriösignaalien monitorointijärjestelmän ohjelmisto sekä sen toiminta moduulitasolla. Luvun alussa esitellään ohjelmiston tuottamisessa käytetyt työkalut niiden asentamisesta lähtien. Luvussa pohditaan lisäksi työssä toteuttamattoman FPGA-ajurin avulla saavutettavissa olevia etuja ja toisaalta FPGA-kehityksen heikkouksia verrattuna tietokoneella toteutettavaan ohjelmistototeutukseen.

5.1 GNU Radio

GNU Radio on signaaliprosessointiin käytettävä ilmainen ohjelmistokehitystyökalu, joka on suunniteltu erityisesti ohjelmistoradioiden yhteydessä käytettävien ohjelmistojen kehittämiseen. GNU Radio:ta jaetaan GNU GPL –lisenssin [78] alaisuudessa, eli sen lähdekoodi on vapaasti saatavilla, mikä yksinkertaistaa sen jatkokehittämistä. GNU GPL –lisenssi kuitenkin rajoittaa jatkokehitetyn ohjelmiston kaupallista käyttöä, sillä lisenssin ns. pysyvyydestä (engl. persistence) ja virusvaikutuksesta (engl. viral effect) johtuen myös koodiin lisätyt tai siihen perustuvat osat tulee julkaista kyseisen avoimen lisenssin alla. GNU Radio:ta käytetäänkin yleisesti tutkimus- ja opetuskäytössä sekä muissa käyttökohteissa, joissa itse ohjelmiston kaupallisuus ei ole keskeisessä asemassa. [78] [69] [79]

GNU Radio perustuu Eric Blossomin vuonna 2001 kirjoittamaan ohjelmistoon, joka sai alkunsa SpectrumWare:n Pspectra-ohjelmiston päälle kehitetystä laajennuksesta. GNU Radio on myöhemmin vuonna 2004 kirjoitettu kokonaan uusiksi, eikä se sisällä enää alkuperäistä Pspectraan perustuvaa koodia. Eric Blossom toimi GNU Radio:n projektin ylläpitäjänä aina vuoden 2010 syyskuuhun asti, minkä jälkeen projektin ylläpitäjänä on toiminut Tom Rondeau. Myös Ettus Research:n perustaja Mark Ettus on ollut osallisena GNU Radio:n kehityksessä jo ohjelmiston hyvin varhaisesta vaiheesta lähtien. [69] [80]

GNU Radio:n ohjelmistokoodin omistaa Free Software Foundation. Ohjelmiston tarkoituksena on tukea radiotaajuisiin ilmiöihin liittyvien innovaatioiden syntymistä tarjoamalla käyttäjälle helposti lähestyttävät työkalut radiotaajuisen spektrin tutkimiseen. GNU Radio tukee seuraavia käyttöjärjestelmiä: Linux, Windows, Mac OS X ja NetBSD. GNU Radio:sta on mahdollista löytää kattavasti tietoa projektin kotisivuilta löytyvän ”Suggested Reading”–linkkikokoelman [81] kautta. [69] [80, pp. 18-26]

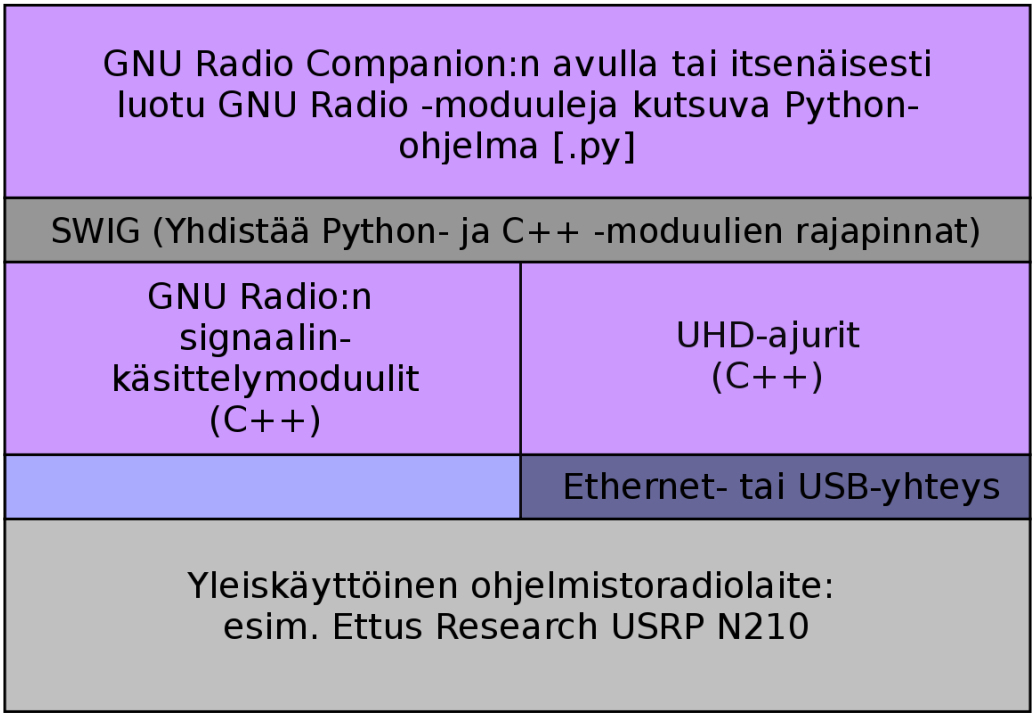
5.1.1 GNU Radio -ohjelmien rakenne

GNU Radio:n avulla kehitettävät ohjelmistot kirjoitetaan usein pääosin Python-ohjelmointikielellä ja sen suoritusnopeuden kannalta kriittiset osat moduuleina C++-ohjelmointikielellä. Käytäntö on peräisin itse GNU Radio:n ohjelmistorakenteesta, sillä se koostuu suuresta kokoelmasta yksittäisiä signaalinkäsittelyn toiminnallisuuksia toteuttavia C++ moduuleja sekä näitä moduuleja kutsuvia ja linkittäviä Python-skriptejä. Edellä kuvattu modulaarinen toimintamalli helpottaa GNU Radio:n yhteensopivuuden ylläpitä-

mistä, tuotetun ohjelmiston osien jatkokehityksen kohdistamista sekä ohjelman eri toiminnallisuuksien testaamista. Korkean tason kielen, kuten Python:n, hyödyntäminen yksinkertaistaa siis järjestelmän jatkokehittämistä sen käyttöliittymän äärimmäisen nopeuden kustannuksella. Ohjelmistokoodin tulkinta ja dataliikenteen ohjaus sen sisältämien C++ ja Python–osien välillä on toteutettu Simple Wrapper Interface Generator:n, eli SWIG:n avulla. SWIG on esitelty tarkemmin luvussa 5.1.4. [82] [83] [69]

GNU Radio ei ohjelmistoprojektina itsessään ota kantaa sillä ohjattavaan laitteistokokoonpanoon, vaan sitä voidaan käyttää hyvin erilaisissa käyttökohteissa käytettävien ohjelmistojen toteuttamiseen. GNU Radio:ta voidaan käyttää ulkoisten mittalaitteiden, kuten ohjelmistoradiolaitteiden kanssa, tai itsenäisesti simulaatioita ja/tai aiemmin nauhoitettua dataa hyödyntävissä ohjelmissa. Ohjelmistoradiolaitteiden hyödyntämiseen tarvitaan tosin GNU Radio:n kanssa yhteensopivat ajurit. GNU Radio:n avoimen rakenteen ansiosta sen kanssa yhteensopivia ajureita on kuitenkin olemassa hyvin monille erilaisille ohjelmistoradiolaitteille. [82] [83] [69]

GNU Radio:n sisäisten osien toimintaa voidaan kuvailla kuvassa 5.1 esitetyn mallin avulla. GRC:n avulla tai itsenäisesti tuotettu Python-koodi kutsuu SWIG:n avulla tuotettujen rajapintojen kautta GNU Radio:n sisältämiä C++-ohjelmointikielellä kirjoitettuja signaalinkäsittelyn suorittavia moduuleja. Signaalinkäsittelyä optimoidaan lisäksi käyttämällä VOLK-kirjastoa [84], joka hyödyntää prosessorikohtaisia käskykantoja koodin suorituksen tehostamiseen. USRP:n UHD-ajureita on mahdollista kutsua myös suoraan Python-koodista SWIG:n välityksellä tai GNU Radio:n moduulien kautta. Ohjelmistoradiolaitteella mitatut näytteet ja tietokoneelta vastaanotetut komennot siirretään Ethernet- tai USB-liitännän yli. UHD-ajurit mahdollistavat siis tietokoneen ja ohjelmistoradion FPGA:n ajurien välisen kommunikoinnin. [69] [85]



Kuva 5.1: GNU Radio:n avulla toteutetun ohjelman yleinen rakenne. Kuva muokattu lähteestä [73, p. 25]

GNU Radio on ensisijaisesti simulaatiotyökalu, jonka sisäinen toiminta perustuu sen moduulien välisiin datavirtoihin. Datavirrat etenevät ennalta määritellyn kokoisissa paketeissa ajoittimen (engl. scheduler) ohjauksessa lähdemoduuleilta kohti nielumoduuleja. Moduuli voi esimerkiksi odottaa, että se vastaanottaa 2048 lukuarvoa, jonka jälkeen se käsittelee kyseisen lukuarvojonon ja lähettää muokkaamansa lukuarvojonon kokonaisena eteenpäin seuraavalle moduulille. Signaalien eteneminen, ja siis koko ohjelman suoritusnopeus, rajoitetaan hitaimman signaalitiellä sijaitsevan moduulin mukaiseksi. Hitaiden moduulien puuttuessa ohjelman suoritusnopeuden määrittävät käytetty näytteistysnopeus sekä ohjelmaa suorittavan järjestelmän suorituskyky. Ohjelman suoritusnopeutta voidaan tosin rajoittaa erillisen ”throttle”-tyyppisen moduulin avulla. Lisäksi käytettyä näytteistysnopeutta voidaan muuttaa ohjelman sisällä, esimerkiksi datavirtoja lomittavien, interpoloivien ja desimoivien moduulien avulla. [69] [86] [87]

GNU Radio:n avulla on mahdollista rakentaa nopeasti ja yksinkertaisesti varsin monipuolisiakin signaalinkäsittelyjärjestelmiä, jos järjestelmän toteuttamiseen tarvittavat moduulit löytyvät valmiina sen kirjastoista. GNU Radio:n ongelmana on kuitenkin sen suhteellisen heikko suorituskyky. Esimerkiksi datan siirtyminen moduulien välillä toteutettu memcpy-funktion avulla, mikä rajoittaa toteutuksen nopeutta. Lisäksi GNU Radio:n rakenteelliset rajoitukset voivat hidastaa uusien moduulien kehittämistä. Järjestelmän sisäisten rajoitusten tunteminen onkin tärkeää, jotta sillä saavutettavat hyödyt voidaan maksimoida. GNU Radio:n modulaarinen rakenne voi esimerkiksi mahdollistaa tietyn järjestelmän rakentamisen hyvinkin lyhyessä ajassa, mutta pahimmillaan se voi toimia pelkästään haittatekijänä, jos järjestelmään täytyy toteuttaa useita uusia moduuleja ja järjestelmältä odotetaan reaaliaikaista toimintaa. [68]

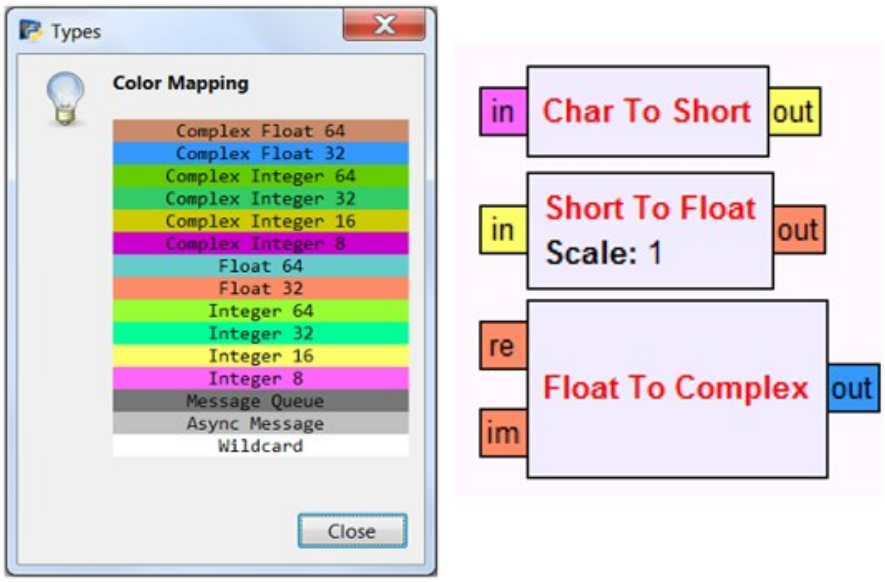
5.1.2 GNU Radio Companion (GRC)

GNU Radio Companion eli GRC sai alkunsa vuonna 2006, kun John Hopkinsin yliopiston silloiset diplomivaiheen opiskelijat Josh Blum ja Patrick Mulligan aloittivat sen kehittämisen professori Brinton A. Cooperin ohjaamassa projektissa, jonka tarkoituksena oli tuottaa graafinen käyttöliittymä GNU Radio:lle. Josh Blum jatkoi tuotetun ohjelmiston kehittämistä vielä projektin jälkeen, mistä johtuen häntä pidetäänkin yleisesti päähahmona GRC:n takana. GRC:tä on käytetty sen kehityksen jälkeen useissa yliopistoissa (Aalto-yliopisto mukaan luettuna) yleisesti signaalinkäsittelyn opetustyökaluna. GRC:tä alettiin jakaa ilmaisohjelmana vuonna 2009 ja se on ollut osa GNU Radio:n normaalijakelua versiosta 3.2 lähtien. [69]

GRC on graafinen käyttöliittymä, joka mahdollistaa GNU Radio:n moduuleja kutsuvan ohjelman tuottamisen graafisesti, ilman erillisen ohjelmakoodin kirjoittamista. GRC toimii seuraavalla tavalla: Käyttäjä luo ohjelmakokonaisuutta kuvaavan signaalivuokaavion (engl. flow graph) ja valitsee GRC:n kirjastosta ohjelmiston toiminnalliset osat muodostavat GNU Radio –moduulit. Tämän jälkeen hän kytkee moduulien sisään- ja ulostulot toisiinsa määritellen näin ohjelman toiminnan. Käyttäjä voi myös määrittää yksittäisissä moduuleissa käytettävät asetukset. Tämän jälkeen käyttäjä voi generoida luotua signaalivuokaaviota vastaavan Python-ohjelmakoodin GRC:n avulla. Luotu ohjelmakoodi on mahdollista käynnistää tämän jälkeen suoraan GRC:stä tai esimerkiksi komentokehotteesta käsin. [73] [69]

GRC tukee hyvin monia GNU Radio:n moduuleja. Esimerkiksi ohjelman signaalilähteinä voidaan käyttää ohjelmistoradion lisäksi erilaisia tietokoneella matemaattisesti luotuja signaaleja, kuten kohinaa tai siniaaltoja; verkon kautta vastaanotettuja TCP- tai UDP-protokollan mukaisia signaaleja; sekä äänikortin sisääntulon kautta- tai binääri- ja wav-tiedostojen avulla luotuja signaaleja. Vastaavasti kyseisiä signaalityyppejä voidaan käyttää ohjelman ulostuloina. Tuettuina on myös moduuleja esimerkiksi signaalien tasojen säätöön, modulointiin, amplitudien-, tehotasojen- ja spektrin mittaukseen, tyyppimuunnoksiin, suodatukseen, virheenkorjaukseen, matemaattiseen operointiin, viivästämiseen sekä synkronointiin. Lisäksi GRC tarjoaa monipuolisesti työkaluja signaalien graafiseen esittämiseen ja ohjelman käyttöliittymän rakentamiseen esimerkiksi erilaisten liukukytkinten ja painikkeiden muodossa. [69] [82] [88]

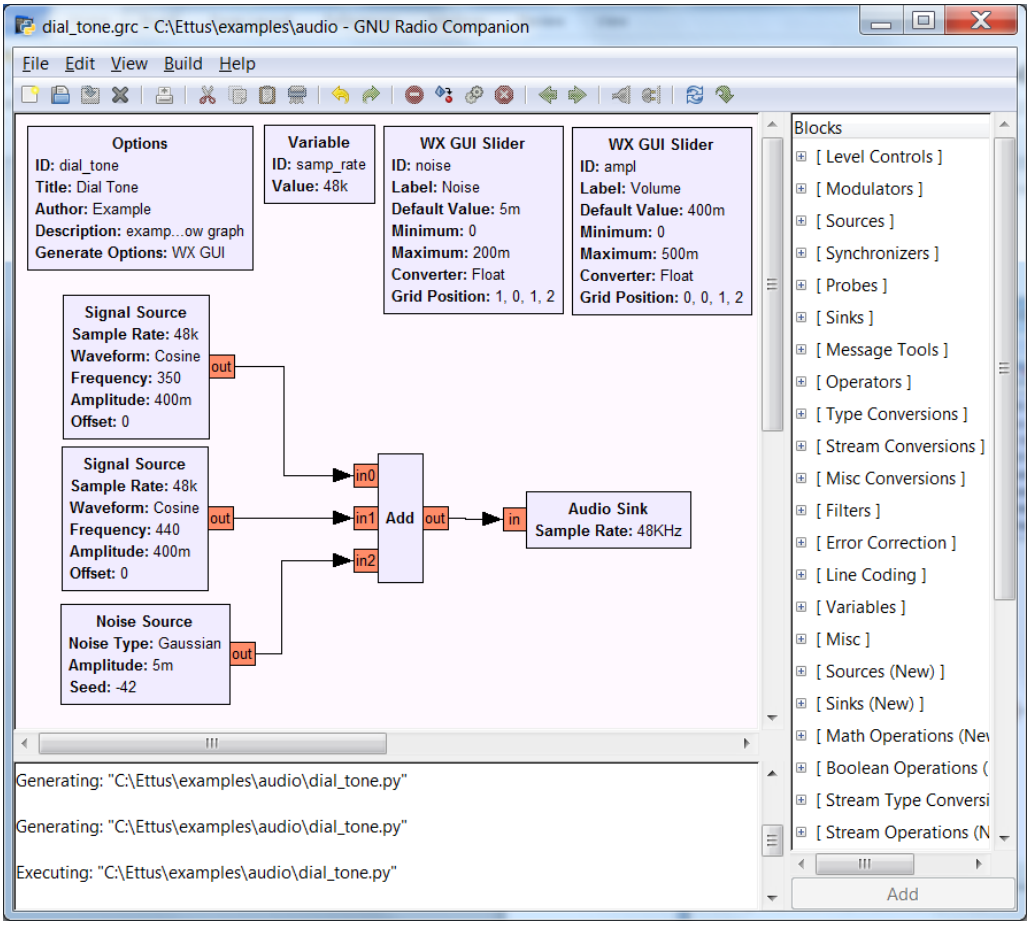
GNU Radio käyttää sisäisesti useita eri datatyyppejä moduulien välisten signaalien esittämiseen. GRC:ssä eri tyyppisiä datavirtoja on selvennetty visuaalisesti värikoodauksen avulla. Värit vastaavat signaalityyppejä GRC:ssä seuraavalla tavalla: violetti esittää tavua, keltainen lyhyttä kokonaislukua, vihreä kokonaislukua, punainen liukulukua ja sininen kompleksilukua. Python-koodissa tietyn moduulin käyttämä datatyyppi taas voidaan yleensä tunnistaa moduulin loppuliitteen (engl. suffix) avulla. Esimerkiksi moduuli ”gr_rms_cf” sisältää loppuliitteen ”_cf”, joka tarkoittaa sitä, että se vastaanottaa 8 tavun pituisia kompleksiarvoisia lukuja ja tuottaa neljän tavun pituisia liukulukuja. Samalla tavalla moduuli ”gr_multiply_const_vss” vastaanottaa kahden tavun pituisen kokonaislukumuotoisen vektorin ja tuottaa ulostulonsa samassa muodossa. GRC:n tärkeimpiä datatyyppejä ja niistä käytettäviä värimerkintöjä on selvennetty kuvassa 5.2. Kuvassa näkyvä ikkuna on mahdollista avata GRC:ssä valitsemalla ohjelman Help-valikon alta löytyvä ”Types”-välilehti. [69]



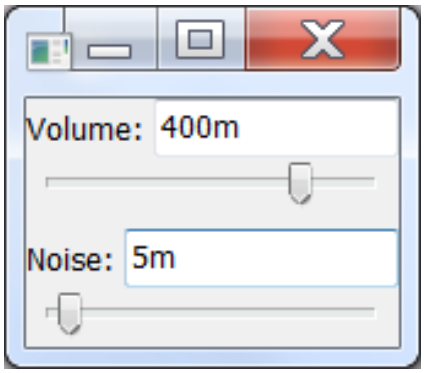
Kuva 5.2: GNU Radiossa käytettävät datatyypit niiden GRC:n värivastineet

Kuva 5.3 esittää GRC:n ohjelmanäkymää, kun avattuna on ”dial tone”–esimerkkitiedosto. Kuvan oikeassa reunassa on nähtävillä valittavissa olevat GNU Radio:n moduulit (Blocks), joita on GRC:n kirjastoissa noin 200 kappaletta. Uusia moduuleja on myös mahdollista luoda itse lisää noudattaen GNU Radio:n kotisivuilta löytyviä ohjeita [89].

Kuvan 5.3 esimerkkiohjelma koostuu kahdesta signaalilähteestä (Signal Source), kohinalähteestä (Noise Source), lähteiden signaalit yhdistävästä summaajasta (Add) sekä summasignaalin toistavasta ääniulostulosta (Audio Sink). Lisäksi ohjelmassa on määritelty erillinen vakio näytteenottotaajuudelle (Variable – ID: samp_rate) sekä liukusäätimet kosinilähteiden ja kohinalähteen amplitudeille. Kuva 5.4 esittää kyseisen esimerkkitiedoston avulla generoidun valmiin ohjelmakoodin ajonaikaista graafista ulkoasua. [69]



Kuva 5.3: GNU Radio Companion:n ohjelmistonäkymä, kun ohjelmalla avattu ”dial_tone.grc”-esimerkkitiedosto



Kuva 5.4: GNU Radio Companion:n avulla generoitu valmis “dial_tone.py”-ohjelma

5.1.3 Python–ohjelmointikieli ja NumPy-kirjasto

Python on korkean tason ohjelmointikieli, jota voidaan käyttää oliopohjaisena, proseduraalisena tai funktionaalisena, dynaamisesti tyypitettynä ohjelmointikielenä. Dynaaminen tyypitys tarkoittaa sitä, ettei koodin avulla luotujen muuttujien tyyppejä tarvitse esitellä erikseen, vaan niiden tyyppi määritellään automaattisesti ohjelman ajon aikana. Python-koodi suoritetaan yleensä virtuaalikoneen avulla, mikä nopeuttaa ja monipuolistaa sen avulla tuotettujen sovellusten testaamista. Suoritettavia ohjelmistorakenteita ei kuitenkaan välttämättä optimoida ennen niiden suoritusta, minkä ansiosta ne voivat olla suoritusnopeudeltaan selvästi hitaampia vastaaviin C/C++-toteutuksiin verrattuna. Python ei siis itsessään sovellu erityisen hyvin operationaalisesti raskaiden ohjelmien toteuttamiseen, mutta se soveltuu erittäin hyvin erilaisten laajennusten ja moduulien kehittämiseen ja hyödyntämiseen. [90] [91, pp. 9-11]

NumPy on alunperin Travis Oliphant:n kehittämiin Numeric- ja Numarray- kirjastoihin perustuva Python-ohjelmistokirjasto, joka on kehitetty tieteellisen laskennan työkaluksi pienentämään Python- ja C/C++-kielten välistä nopeuseroa. NumPy pyrkii toimimaan ilmaisena, avoimeen lähdekoodiin perustuvana vaihtoehtona kaupalliselle MATLAB-ohjelmistolle. NumPy:n sisäinen rakenne on toteutettu Python- ja C-ohjelmointikielillä ja sen tehokkuus perustuu moniulotteiseen ”ndarray”-taulukkorakenteeseen ja rakenteen manipulointiin kehitettyihin C++-ohjelmointikielellä toteutettuihin tehokkaisiin operointifunktioihin. [91, pp. 9-11] [92]

5.1.4 SWIG

SWIG on käyttöliittymän ohjelmistokoostaja, jonka avulla on mahdollista luoda rajapinta C/C++-ohjelmointikielellä tuotettujen rakenteiden sekä skriptipohjaisten ohjelmistojen välille, niiden keskinäisen kommunikoinnin mahdollistamiseksi. SWIG tuottaa toiminnallisuuden linkittävän koodin C/C++-ohjelman otsikkotiedoston pohjalta. Näin luotu linkkikoodi mahdollistaa C/C++-ohjelman rakenteiden kutsumisen suoraan skriptipohjaisesta, esimerkiksi Perl-, Python-, Ruby- tai TCL- tyyppisestä ohjelmakoodista. GNU Radio:ssa SWIG:n käyttö mahdollistaa nopeiden signaalinkäsittelymoduulien luomisen samalla ylläpitäen skriptipohjaisella Python-ohjelmointikielellä saavutettavat ohjelmistokehitykseen liittyvät edut. Lisäksi käytäntö yksinkertaistaa uusien C/C++-signaalikäsittelymoduulien luomista. [83] [93]

SWIG on ollut vapaasti saatavilla helmikuusta 1996 lähtien ja projektiin on osallistunut suuri määrä ulkoisia ohjelmistokehittäjiä. Nykyään SWIG:n ylläpito onkin kokonaan vapaaehtoistyön varassa. SWIG on saatavilla esim. Github:sta [94], joskin suurin osa Linux-distribuutioista sisältää SWIG:n jo normaalijakelussaan. Ohjelman version päivittämistä kuitenkin suositellaan, sillä normaalijakeluiden versiot eivät välttämättä päivity automaattisesti. [93]

5.1.5 Python-ohjelmointi GNU Radio:n avulla

GNU Radio:n lähde-, nielu- ja signaaliprosessointimoduulien sisäiset rakenteet on toteutettu C++-moduulien avulla ja näitä moduuleja voidaan kutsua suoraan Python-ohjelmointikielisestä ohjelmasta SWIG:llä toteutettujen rajapintojen kautta. Modulaarinen rakenne edesauttaa ohjelman toimintaa ohjaavan Python-koodin helppolukuisuuden säi-

lyttämisessä toteutetun ohjelmiston monimutkaisuudesta riippumatta [83]. Kuvassa 5.5 on esitetty GNU Radio:n toimintaa kuvaava ”Hello World” -tyyppinen ohjelma. Ohjelmassa luodaan yksinkertainen 440 Hz:n sinisignaali, jonka jälkeen se yhdistetään ääniulostuloon eli soitetaan tietokoneen äänikortin kautta.

```
#!/usr/bin/env python
from gnuradio import analog
from gnuradio import audio
from gnuradio import gr

class tone(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self)
        self.samp_rate = samp_rate = 48000
        self.ampl = ampl = .4
        self.audio_sink = audio.sink(samp_rate)
        self.src0 = analog.sig_source_f(samp_rate,
                                         analog.GR_SIN_WAVE,
                                         440, ampl)
        self.connect((self.src0, 0), (self.audio_sink, 0))

if __name__ == '__main__':
    tb = tone()
    tb.start()
    raw_input('Press any key to quit')
    tb.stop()
```

Kuva 5.5: Python-ohjelmointikielellä toteutettu GNU Radio -esimerkkiohjelma

GNU Radio –ohjelmat seuraavat yleisesti kuvassa 5.5 esitettyä rakennetta. Itse ohjelman signaalinkäsittely suoritetaan ”tone” -nimisessä, ”gr.top_block”–moduulista perivässä, ohjelman vuokaaviona toimivassa luokassa. Ohjelmassa käytetyt ”samp_rate”- ja ”ampl” -vakiot määritellään luokan sisällä. Ääniulostulo taas muodostetaan ”audio.sink()”-funktion avulla. Kyseinen funktio ottaa sisääntuloarvoikseen käytettävän näytteenottotaajuuden sekä ääniulostulona toimivan laitteen nimen. Käytettävän laitteen nimeä ei ole määritelty koodissa, joten ohjelma käyttää funktion oletusarvoa vastaavaa laitetta. Kyseinen äänինieluna toimiva moduuli vastaanottaa yhden tai useampia liukulukuvirtoja ja lähettää luvuista luodun signaalin edelleen tietokoneen äänikortille. Ulostulon määrittelyn jälkeen koodissa generoidaan soitettava siniaalto ”analog.sig_source_f”–funktiokutsun avulla. Funktion ”_f”-päätteestä voidaan nähdä, että kyseinen lähteenä toimiva moduuli tuottaa liukuluku-tyyppisen ulostulosignaalin. Funktio ottaa sisääntuloarvoikseen näytteenottotaajuuden, signaalin tyypin, signaalin taajuuden sekä amplitudin. Siniaalto generoidaan siis 440 Hz:n taajuudella ja amplitudilla ”0.4”. Moduulit yhdistetään toisiinsa ”connect”–funktion avulla. Kyseisen funktion toiminnan määrittelevät parametrit ovat sen käyttämät lähteen ja nielun päätepisteet. Nielun päätepiste koostuu signaaliprosessointimoduulista sekä ulostuloporttien lukumäärästä. Yleisessä muodossa päätepiste on määritelty muodossa (moduuli, portin numero).

Itse pääohjelmassa määritellään luotua vuokaaviota kuvaavaa ”tone”-luokkaa vastaava muuttuja ”tb”, jonka jälkeen ohjelma kutsuu sen ”start”-funktiota. Kyseinen funktio

käynnistää prosessisäikeen vuokaavion suorittamiseen ja luovuttaa kontrollin välittömästi takaisin kutsujalle. Kyseisessä ohjelmassa koodin suorittamista jatketaan kunnes käyttäjä painaa mitä tahansa painiketta. Painalluksen jälkeen ohjelma pysäyttää vuokaavion suorituksen kutsumalla sen ”stop”-funktiota.

5.1.6 GNU Radio:n asentaminen

GNU Radio käyttää toiminnoissaan hyvin laajaa valikoimaa erilaisia ulkoisia ohjelmistokirjastoja. Tarvittavien kirjastojen täytyykin olla asennettuna tietokoneelle ennen kuin niitä hyödyntäviä GNU Radio –moduuleja voidaan käyttää. GNU Radio:n koostaminen ja asentaminen sen lähdekoodista lähtien voi olla varsin epätriviaali operaatio, jos asennus suoritetaan muutenkin kyseisten kirjastojen aiempia, epäyhteensopivia versioita hyödyntävälle käyttöjärjestelmälle. Uudemmat Linux-pohjaiset käyttöjärjestelmät sisältävät tarvittavat kirjastot jo niiden normaalijakelussa, mikä voi yksinkertaistaa GNU Radio:n asentamista. Parhaassa tapauksessa normaalijakelun sisältämä GNU Radio:n versio voidaankin asentaa suoraan käyttämällä järjestelmien normaaleja asennustyökaluja, eli esimerkiksi Ubuntu:ssa ja Debian:ssa komennolla: [69] [95]

```
$ apt-get install gnuradio
```

Linux-pohjaisten käyttöjärjestelmien normaalijakeluiden sisältämät GNU Radio versiot voivat kuitenkin olla ohjelmiston nopeasta kehityksestä johtuen jo vanhentuneita. GNU Radio:n kotisivuilla kehotetaankin asentamaan aina ohjelmiston uusin versio, sillä sivustolta saatava tuki kohdistetaan vain siihen. Uusin versio GNU Radio:sta ja sen tarvitsemista kirjastoista on mahdollista asentaa esimerkiksi käyttämällä Marcus Leechin kehittämää, GNU Radio:n asentamisesta huolehtivaa ”build-gnuradio”-nimistä Python-ohjelmaa. Kyseinen ohjelma voidaan ladata ja asentaa komennoilla: [69]

```
$ wget http://www.sbrac.org/files/build-gnuradio && chmod a+x ./build-gnuradio && ./build-gnuradio
```

GNU Radio ei ole virallisesti tuettuna Windows-käyttöjärjestelmässä, minkä takia sen käyttämiseen suositellaankin Linux-pohjaisia käyttöjärjestelmiä. Windows käyttöjärjestelmässä voidaan tosin käyttää Corgan Labs:n koostamaa ”GNU Radio Live SDR” nimistä GNU Radio:n versiota, joka on DVD:lle, USB-muistille tai esimerkiksi virtuaalikoneelle asennettavissa oleva Ubuntu GNOME:n päälle rakennettu kokonainen käyttöjärjestelmäkuva. GNU Radio Live SDR sisältää valmiiksi asennettuna kaikki GNU Radio -ohjelmiston käyttämiseen tarvittavat kirjastot. GNU Radio Live SDR on saatavissa lähteestä [96].

GNU Radio on mahdollista asentaa myös suoraan Windows-käyttöjärjestelmään seuraamalla Ettus Research:n sivuilta löytyviä asennusohjeita [97]. Asennusohjeet ovat nykyään varsin suoraviivaiset ja yksityiskohtaiset, joten niiden tarkempi läpikäyminen ohitetaan. Huomion arvoista on kuitenkin asentaa tarvittavat kirjastot ja UHD ennen itse GNU Radio:n asentamista. UHD-ajurista tulisi myös olla asennettuna vähintään 3.0.0 –versio, jotta sen toiminta 3.7.x -versiollisten GNU Radio ohjelmistojen kanssa voitaisiin varmistaa. Vastaavasti itse ohjelmistoradiojärjestelmään tulisi olla asennettuna käytössä olevaa UHD-versiota vastaavat FPGA-ajurit. [95]

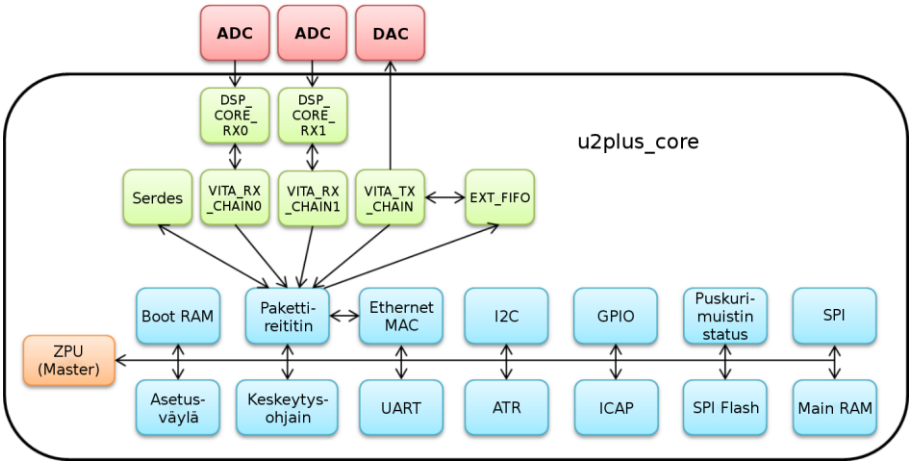
5.1.7 GNU Radio versiomuutokset

GNU Radio:n versioinnissa on pyritty säilyttämään yhteensopivuus sen avulla tuotettujen sovellusten välillä aina kokonaisen versioperheen sisällä. Tällä tarkoitetaan siis sitä, että esimerkiksi versiolla 3.5.1 tuotettu GNU Radio –ohjelma on yhteensopiva myös GNU Radio:n 3.5.2 –version kanssa. GNU Radio:n kehitys on kuitenkin varsin nopeaa ja suurempia koko versioperheen uudistavia päivityksiä voidaankin odottaa ilmestyvän vähintään vuoden välein [69]. Ohjelmistokehityksen kannalta tämä tarkoittaa sitä, että uusien ominaisuuksien hyödyntäminen vaatii myös tuotetun ohjelmiston ylläpitämistä ja päivittämistä vastaamaan käytettyä GNU Radio:n versiota.

GNU Radio:n suurimmat 3.4.0 -version jälkeiset muutokset voidaan tiivistää yleisesti seuraavalla tavalla. Versiossa 3.5.0 ohjelman koostamistyökalut muutettiin autotools–työkalusta Cmake-työkaluun, millä parannettiin järjestelmän yhteensopivuutta Windows-käyttöympäristön kanssa. Myös VOLK-optimisaatiot otettiin käyttöön versiossa 3.5.0 ensimmäistä kertaa GNU-Radio:n historiassa, mikä paransi järjestelmällä saavutettavaa suorituskykyä huomattavasti. Versio 3.6.0 muutti GNU Radio:n moduulien välisten viestien toimintaa, mahdollisti Python-ohjelmointikielellä kokonaan toteutettujen signaalikäsittelymoduulien käytön sekä toi uusia ohjelman suorituskykyä mittaavia toiminnallisuuksia. Versio 3.7.0 yhtenäisti GNU Radio:n lähdekoodin osien keskinäistä rakennetta ja toi uudistuksia moduulien rekursiiviseen ohjaamiseen. Lisäksi versio laajensi suorituskykyä mittaavien työkalujen toimintaa entisestään. Jokainen edellä mainituista versioperheistä sisälsi myös huomattavia päivityksiä GNU Radio:n sisäiseen rakenteeseen ja moduuleihin liittyen. [69]

5.2 FPGA:n ohjelmointi

USRP N210:n sisältämä FPGA-piiri on sen keskeisin yksittäinen osa, sillä sitä käytetään laitteen signaaliprosessoinnissa, aikasignaalien jakamisessa sekä laitteen yhteyksien määrittelyssä. FPGA:n oletusajurit sisältävät 32-bittisen RISC prosessorin (ZPU), jota käytetään laitteen sisäisten toimintojen ohjaamisessa ”Wishbone”-nimisen väylän avulla. FPGA:n ydintoimintoja kuvaavan ajurin sisäinen rakenne on esitetty kuvassa 5.6. Laitteen FPGA-ajurien Verilog-koodit on löydettävissä lähteestä [98].



Kuva 5.6: USRP N210 FPGA:n ydintoimintoja ohjaavan u2plus_core.v –ajurin sisäinen toiminta. Kuva muokattu lähteestä [99, p. 27]

USRP:n sisältämät FPGA-laiteajurit sijaitsevat ”u2plus_core.v” –nimisessä Verilog-tiedostossa. Teoriassa USRP:n FPGA vain vastaanottaa ADC:n näytteet ja lähettää ne eteenpäin tietokoneelle Ethernet-väylän välityksellä. Todellisuudessa näytteistyksen ja tietojen lähettämisen välillä on kuitenkin monia varsin monimutkaisia välivaiheita. Pääpiirteissään näitteistystä voidaan kuvata seuraavalla tavalla: FPGA vastaanottaa ADC:n 100 MSps nopeudella näytteistämät 14-bittiset etumerkilliset kompleksiarvoiset näytteet, jotka vastaanotetaan ”rx_frontend”–moduulissa. Moduuli suorittaa signaaleille jännitteen tasavirtaosuuden biasoinnin (engl. DC-offset) sekä magnitudin- ja vaiheenkorjauksen, joiden avulla signaaleja voidaan synkronisoida useamman USRP laitteen välillä MIMO-konfiguraatiossa. Moduuli sisältää myös tytärkorttien kalibrointiin liittyviä toiminnallisuuksia, jotka vähentävät datavirran häiriöiden määrää. ”rx_frontend”-moduulin jälkeen datavirta etenee ”dsp_core_rx”-moduuliin, jossa sille suoritetaan tietokoneella määritellyt suodatus- ja desimaatioalgoritmit . Moduulin jälkeen data tallennetaan FPGA:n FIFO-linjoihin, joista se lähetetään eteenpäin tietokoneelle Ethernet-yhteyden välityksellä. [77]

USRP N210 –ohjelmistoradiolaite toimii oletusajureillaan vain tietokoneella ohjattavana ulkoisena näytteistyslaitteena ja itse signaalien prosessointi suoritetaan siis vasta järjestelmää ohjaavalla tietokoneella. Näytteiden prosessointi tietokoneen yleiskäyttöisen prosessorin avulla on kuitenkin hidasta, mikä rajoittaa järjestelmällä reaaliaikaisesti suoritettavissa olevien algoritmien monimutkaisuutta. Järjestelmän suorituskykyä voidaankin parantaa käyttämällä esimerkiksi laitteen sisäistä FPGA-piiriä myös itse signaalien käsittelyssä. Laitteen FPGA-piirin ajuri voidaan ohjelmoida VHDL- ja Verilog-laitteistokuvauskielten, sekä nykyisin myös LabVIEW:n ja Simulink:n avulla [63]. Laitteen FPGA:n oletusajurien sisäisen toiminnan ymmärtäminen on kuitenkin tarpeellista, jos järjestelmää lähdetään jatkokehittämään laitteen oletusajurien päälle. Lisäksi tehdyt muutokset voivat aiheuttaa järjestelmän putoamisen sen tarvitsemien aikavakioiden sisältä, mikä hidastaa järjestelmällä saavutettavaa maksimaalista kellotaajuutta ja pahimmassa tapauksessa estää koko järjestelmän toiminnan. [68] [77]

Useissa tutkimuksissa on pyritty välttämään edellä mainittuja ongelmia suorittamalla signaalinkäsittely ulkoisten FPGA-kehitysalustojen avulla. Menettely parantaa samalla järjestelmän päivitettävyyttä, sillä se mahdollistaa laitteen näytteistyspuolta nopeammin kehittyvän signaalikäsittelyosan päivittämisen erillään muusta järjestelmästä. Ulkoisen FPGA-kehitysalustan käyttöä on tutkittu esimerkiksi CRUSH–projektissa [77], jossa USRP:n signaalien jatkokäsittely suoritettiin Xilinx Virtex-6 ML605–kehitysalustan avulla. Projektissa kehitettiin erillinen, USRP:n MIMO-liitäntää hyödyntävä, liitäntäkortti USRP:n ja ulkoisen FPGA-kehitysalustan keskinäisen kommunikoinnin mahdollistamiseksi. Projektissa tehtiin USRP:n alkuperäisiin ajureihin vain mitattujen signaalien ohjaukseen liittyviä pieniä muutoksia, mikä mahdollisti ohjelmistoradion alkuperäisen toiminnallisuuden säilyttämisen. Järjestely mahdollisti tuotettujen FPGA-pohjaisten algoritmien ja prosessoripohjaisen ohjelmistototeutuksen monipuolisen keskinäisen vertailun. [77]

CRUSH-projektissa saavutettiin lyhyillä FFT:n pituuksilla suuria, jopa monisatakertaisia nopeusetuja verrattuna tietokoneella suoritettavaan ohjelmistototeutukseen. Huomion arvoista kyseisessä projektissa saaduissa tuloksissa on kuitenkin se, ettei tietokoneella suoritettavan toteutuksen nopeutta rajoittanut käytetyn tietokoneen prosessointiteho vaan mittausdatan siirrossa käytetyn Ethernet-yhteyden hitaus. Pidemmillä FFT:n

pituuksilla FPGA:n avulla saavutettu nopeusetu pienenikin huomattavasti ja esimerkiksi 4096-alkion pituisella FFT:llä saavutettiinkin enää 8-kertainen nopeusetu verrattuna tietokoneella suoritettuun ohjelmistototeutukseen. [77]

5.3 Toteutettu häiriömonitorointiohjelmisto

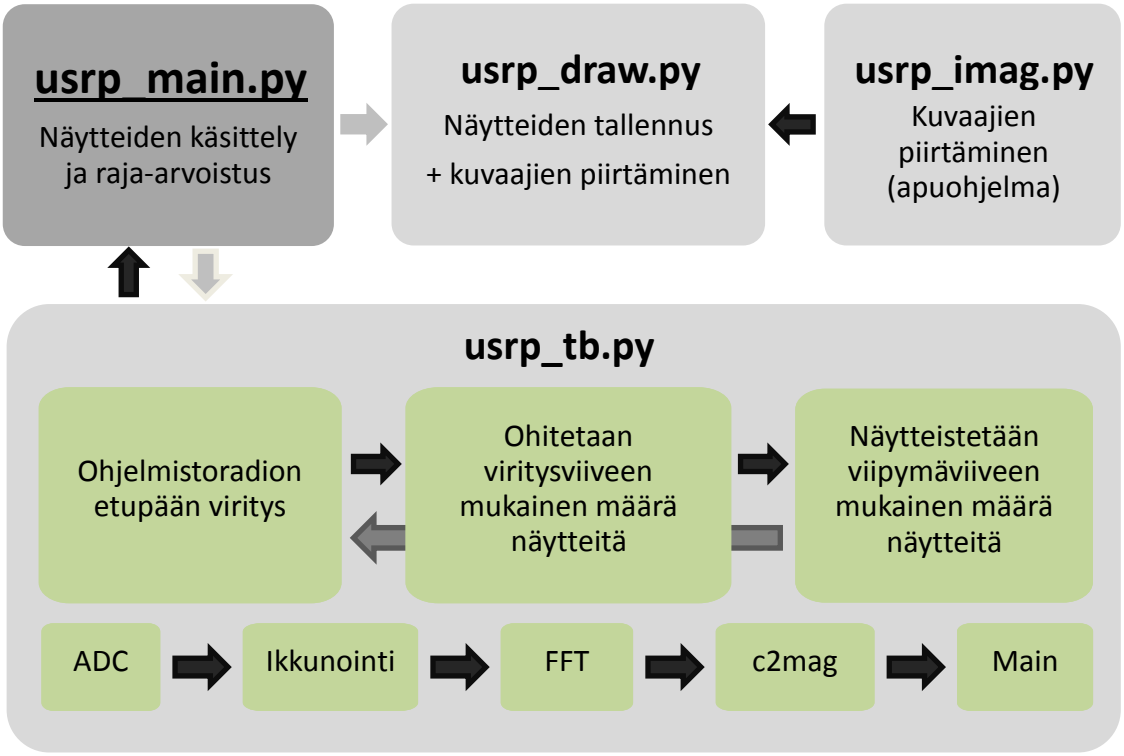
USRP N210 –ohjelmistoradiolaite toimii siis oletusajureillaan vain ulkoisena, ohjattavana näytteistyslaitteena. USRP alasmuuntaa, näytteistää ja siirtää näytteet Ethernet-yhteyden välityksellä tietokoneelle, jossa ne vastaanotetaan UHD-ajurien avulla. Vastaanotettua dataa voidaan tämän jälkeen prosessoida tietokoneessa esimerkiksi GNU Radio:n ja Python-koodin avulla. Tässä työssä toteutettu ohjelmisto (liitteet 3-7) perustuu GNU Radio:n moduuleilla toteutettuun, laajakaistaisen spektrin analysoinnin mahdollistavaan ”usrp_spectrum_sense.py”–esimerkkiohjelmistoon [100], joka koostuu erillisestä ohjelman toiminnan määrittelevästä ”gr.top_block”-luokasta sekä tätä kutsuvasta pääohjelmasta. Lisäksi se sisältää ohjelman muita toimintoja ohjaavia luokkia, kuten laitteen tytärkortin virityksestä huolehtivan ”gr.feval_dd”-tyyppisen ”tune”-luokan sekä mitattujen näytteiden jäsentelystä huolehtivan ”parse_msg”-luokan. USRP N210 mahdollistaa korkeintaan 25 MHz:n kaistanleveyden reaaliaikaisen näytteistämisen, joten laajakaistaisen spektrin analysointi suoritetaan jakamalla spektri sopiviin osiin ja virittämällä laitteen etuastetta valittujen osien mukaisesti. Laite toimii siis hybridispektrianalysaattorina (luku 2.6).

Toteutetussa häiriömonitorointiohjelmistossa edellä mainittua esimerkkiohjelman toiminnallisuutta on laajennettu lähinnä laitteen etupään virityksestä huolehtivien-, sekä ohjelman sisäisiä asetuksia ohjaavien funktioiden osalta. Näin saatu kokonaisuus on tallennettu ”usrp_tb.py” –tiedostoon, joka siis toimii pääohjelmasta kutsuttavana, USRP N210 ohjelmistoradion toimintaa ohjaavana kokonaisuutena. Itse pääohjelma on toteutettu ”usrp_main.py”–tiedostossa. Myös muut ohjelman selkeät erilliset kokonaisuudet on jaettu omiin tiedostoihinsa ohjelmiston rakenteen modularisoimiseksi. Pääohjelman toimintaan liittyvät ohjelmiston osat on esitetty taulukossa 5-1 ja niiden lähdekoodit on esitetty liitteissä 3-7.

Tiedostonimi	Kuvaus
usrp_main.py	Pääohjelma, joka toteuttaa mitattujen signaalien tasojen vertailun, tiedostojen tallentamisen sekä muut ohjelman ydintoiminnot
usrp_tb.py	USRP N210:n toimintoja ohjaileva kirjasto
usrp_conf.py	Ohjelmiston konfiguraatioista huolehtiva kirjasto
usrp_draw.py	Mitattujen signaalien piirtofunktiot sisältävä kirjasto
usrp_imag.py	Ulkoinen ohjelma piirrettyjen kuvien koostamiseen
usrp_zeroes.py	Kalibrointiohjelma laitteen häiriöiden minimointiin
usrp_calibration.py	Kalibrointiohjelma laitteen taajuusvasteen linearisointiin

Taulukko 5-1: Toteutetun ohjelmiston pääohjelmaan liittyvät kokonaisuudet

Toteutetun häiriömonitorointiohjelmiston peruspiirteinen toiminta on esitetty kuvassa 5.7. Ohjelman toiminta voidaan jakaa seuraaviin osiin: Ohjelman käynnistyessä sen päätoiminnallisuuden toteuttava ”usrp_main.py”-pääohjelma käynnistää itse ohjelmistoradiota ohjaavan ”usrp_tb.py”-aliohjelman, jota suoritetaan tämän jälkeen erillisessä prosessisäikeessä rinnakkain itse pääohjelman kanssa. ”usrp_tb.py” myös lataa ohjelman konfiguraatiotiedot "config.txt"-tiedostosta. Lisäksi asetuksia voidaan määritellä ohjelman käynnistyksen yhteydessä kutsuttavien komentokehote-argumenttien avulla. Asetusten avulla voidaan muuttaa esimerkiksi näytteistettävää taajuusaluetta, käytettävää näytteenottotaajuutta sekä käytettyjä viritys- ja viipymäviiveitä. Pääohjelma luo ohjelman asetusten lataamisen jälkeen jälkeen nykyistä päivämäärää vastaavat tiedostokansiot datan ja luotujen kuvien tallennusta varten, minkä jälkeen ohjelma siirtyy normaaliin toimintamoodiinsa.



Kuva 5.7: Häiriömonitorointiohjelman pääpiirteinen toiminta

Normaalissa toimintamoodissa ohjelmisto käy läpi seuraavat vaiheet: ”usrp_tb.py”-apuohjelma lähettää ohjelmistoradiolle käskyn virittää sen vastaanottimena toimivan tytärkortin etuaste halutulle taajuudelle. Käskyn lähettämisen jälkeen ohjelma hylkää ennalta valitun viritysviiveen avulla lasketun lukumäärän ohjelmistoradiolta vastaanotettuja näytteitä. Viritysviiveen tehtävänä on varmistaa, että ohjelmistoradio on vastaanottanut sille lähetetyt käskyt, sen etuaste on viritettynä halutulle taajuudelle ja että laitteen sisäiset FIFO-puskurit on saatu täytettyä kyseistä keskitaajuutta vastaavilla uusilla näytteillä. Tämän jälkeen apuohjelma tallentaa muistiinsa viipymäviiveen (engl. dwell delay) avulla määritellyn ajan kompleksimuotoisia 32-bittisiä I/Q-näytteitä, jotka muunnetaan taajuustasoon Fourier’n muunnoksen avulla. Tämän jälkeen mitatut FFT-lokerot neliöllistetään, mikä muuttaa ne reaaliarvoisiksi. Näin saadut arvot lähetetään eteenpäin ohjelman ydintoimintoja ohjailevalle ”bin_statistics_f”-moduulille. Kyseinen moduuli sisältää tilakoneen mittaustietojen vastaanottamiseen sekä laitteen etuasteen virittämiseen.

Moduuli säästää viipymäviiveen aikana näytteistetyt maksimiarvoiset FFT-lokerot, minkä jälkeen se lähettää näistä koostetun viestin eteenpäin pääohjelmalle. Viipymäviiveen avulla voidaan siis määrittää kuinka monta kokonaista, valitun pituista FFT-funktiota lasketaan jokaisen viritysaskeleen sisällä. Viritysviiveen ollessa pienempi tai yhtäsuuri kuin yhden FFT:n näytteistämiseen kuluva aika, näytteistetään vain yksi FFT, jolloin maksimiarvoistusta ei suoriteta. Tietojen lähetyksen jälkeen laitteen etupää viritetään uudelleen seuraavan näytteistettävän taajuuskaistan keskitaajuudelle ja näytteistyskierros aloitetaan alusta.

Pääohjelmassa vastaanotettuja FFT-lokeroiden arvoja verrataan ennalta asetettuun raja-arvoon ja raja-arvon ylittävät taajuudet, mittausajankohdat sekä mitatut amplitudit tallennetaan tekstitiedostoon. Lisäksi taajuusalueen FFT-lokeroiden maksimi-, minimi- ja keskiarvot tallennetaan erilliseen Numpy-taulukkoon. Valitun pyyhkäisymäärän suorituksen jälkeen ohjelma tallentaa kyseisen taulukon kiintolevylle. Näin saaduista mittaustuloksista voidaan tämän jälkeen luoda spektri- ja spektrogrammikuvaajia tietyin aikavälein itse pääohjelman, tai erillisen ”usrp_imag.py”-apuohjelman avulla, kutsuen ”usrp_draw.py”-kirjastossa toteutettuja piirtofunktioita.

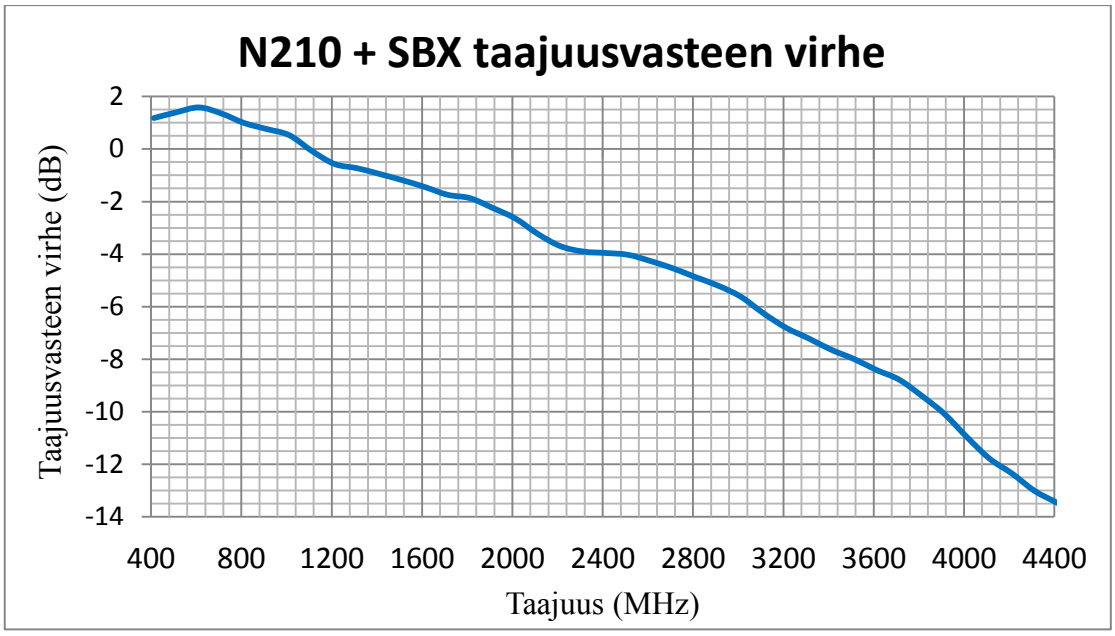
5.4 Järjestelmän rajoitukset, häiriöt ja kalibrointi

Vertailukelpoisten absoluuttisten mittausten mahdollistaminen työssä käytetyn järjestelmän avulla asettaa monia vaatimuksia toteutettavalle ohjelmistolle. Esimerkiksi USRP N210:n mittaamia jännitearvoja ei ole laitteen oletusajureissa kiinnitetty mihinkään tunnettuun referenssiin, mikä tekee järjestelmän kalibroinnin välttämättömäksi. Vastaavasti koko monitorointijärjestelmän taajuusvaste on sen yksittäisten osien, kuten antennin, kaapeloinnin, vahvistimien sekä ohjelmistoradion tytärkortin taajuusvasteiden summa. Jos mitatut signaalitasot halutaan siis muuntaa mitatuksi sähkövuon tehotiheydeksi, tulee edellä mainittujen osien aiheuttama vahvistus (tai vaimennus) ottaa huomioon. [69]

USRP sisältää myös useita sen komponenttien epäideaalisuuksista, kuten epälineaarisuudesta, näytteistyksen kvantisaatiotasojen rajallisuudesta, sekä vaihe- ja ryhmäviiveistä ja näiden keskinäisistä eroista johtuvia virhelähteitä. Yleisimmät USRP:n virhelähteet, niiden aiheuttamat virhetyypit, sekä virheiden kompensoinnin perusperiaatteet on esitetty lähteessä [101]. Järjestelmän linearisointi ja virhesignaalien minimointi vaatii järjestelmän komponenttien tuntemista. Käydään seuraavaksi läpi toteutetun ohjelmistoradiojärjestelmän kalibrointiin sekä sen muutamien virhetekijöiden karakterisointiin ja niiden vaikutusten minimointiin käytettyjä menetelmiä. [69]

5.4.1 Järjestelmän taajuusvasteen kalibrointi

Työssä käytetylle Ettus Research USRP N210 ohjelmistoradiolle toteutetun ohjelmiston kanssa käytettiin etuasteena SBX-tytärkorttia, sillä sen 400 – 4400 MHz:n taajuusalue kattaa suurimman osan radioastronomisten vastaanotinten IF-välikaistoille osuvista häiriöistä. USRP N210:n SBX-tytärkortin taajuusvasteen virhe mitattiin käyttämällä Rohde & Schwarz:n SMB100A -signaaligeneraattoria ja GRC:n ”uhd_FFT”-esimerkkiohjelmaa. Tytärkortin 400 – 4400 MHz:n taajuusalueesta pyyhkäistiin 25 MHz:n kaista 100 MHz:n välein ja jokaiselta mittauskaistalta otettiin talteen kolme eri amplitudiarvoa vasteen lineaariselta alueelta. Tytärkortin vahvistus asetettiin arvoon 0 dB. Mitattu taajuusvasteen virhe on esitetty kuvassa 5.8.



Kuva 5.8: Ettus Research N210 + SBX tytärkortin 400 - 4400 MHz taajuusalueen taajuusvasteen virhe

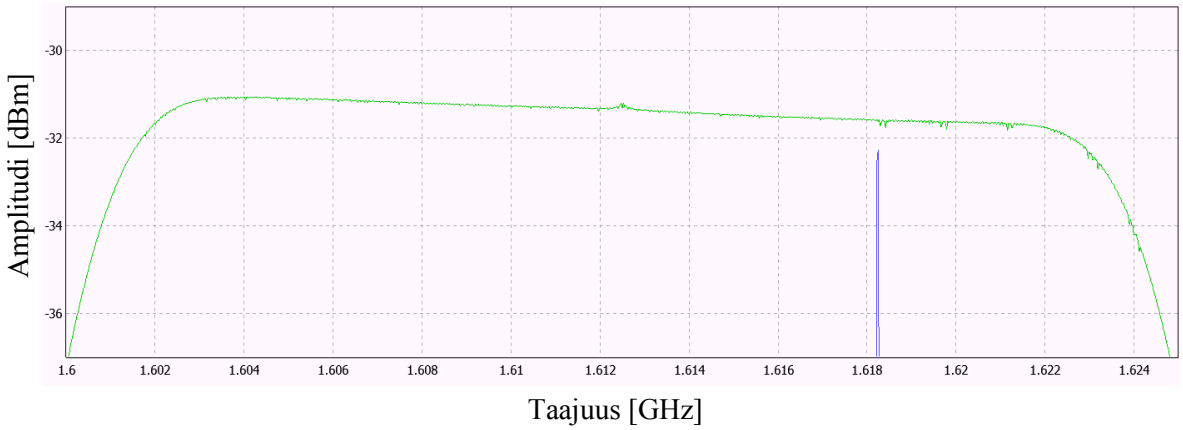
Saadut mittaustulokset mahdollistavat mitattujen signaalien ohjelmallisen kalibroinnin. Työn aikana toteutettiinkin useita menetelmiä mitattujen signaalien kalibroimiseksi. Kalibrointi suoritettiin aluksi ohjelmallisesti kertomalla mitatut signaalit matemaattisesti edellä kuvattujen mittausten pohjalta tuotetun kalibrointitaulukon avulla. Lopullisessa ohjelmistossa kalibrointi suoritetaan kuitenkin jo mittausten aikana muuttamalla laitteen etuasteen vahvistimien vahvistusta viritettävän keskitaajuuden funktiona. Menettely poistaa yhden ylimääräisen signaaliprosessointivaiheen ja samalla se mahdollistaa laitteen näytteistyskomponenttien dynaamisen alueen tehokkaamman hyödyntämisen.

Laitteen kalibrointiarvot tulisi kuitenkin mitata aina uudelleen, jos käytettyä näytteistyskaistan leveyttä, LO-offset:n arvoa tai muita laitteen asetuksia muutetaan. Työssä toteutettiinkin alustavasti erillinen kalibrointiohjelma, jolla pyrittiin yksinkertaistamaan järjestelmän kalibrointia. Kalibrointiohjelman toiminta perustuu ulkoisen signaaligeneraattorin käyttämiseen ohjelmistoradiolla mitattujen tehotasojen referenssinä.

Kalibrointiohjelman toiminta voidaan jakaa seuraaviin osiin. Ohjelma vaihtaa signaaligeneraattorilla pyyhkäistävän taajuuskaistan vastaamaan kalibroitavan taajuusalueen yksittäistä viritysaskelta. Tämän jälkeen myös ohjelmistoradiolaitteen etupää viritetään kyseiselle keskitaajuudelle, minkä jälkeen taajuusaluetta näytteistetään, kunnes sitä vastaavien FFT-lokeroiden arvot ylittävät tietyn ennalta määrätyn raja-arvon. Tämän jälkeen käytettyjä taajuusalueita muutetaan, kunnes koko taajuusalue tulee kalibroiduksi.

Ilman ulkoista ohjausta suoritettava järjestelmän kalibrointi on kuitenkin varsin hidas prosessi. Kalibrointijärjestelmän toimintaa yritettiinkin tehostaa työn aikana ohjaamalla myös kalibroinnissa käytettyä signaaligeneraattoria SCPI-komentojen avulla. Komentojen hyödyntämiseen toteutettiin ”smb100A.py”-ohjelmistokirjasto, jonka toiminta perustuu National Instruments:n VISA-ajurin hyödyntämiseen. Järjestelmää ei kuitenkaan saatu toimimaan vakaasti lähinnä yhteysongelmista johtuen.

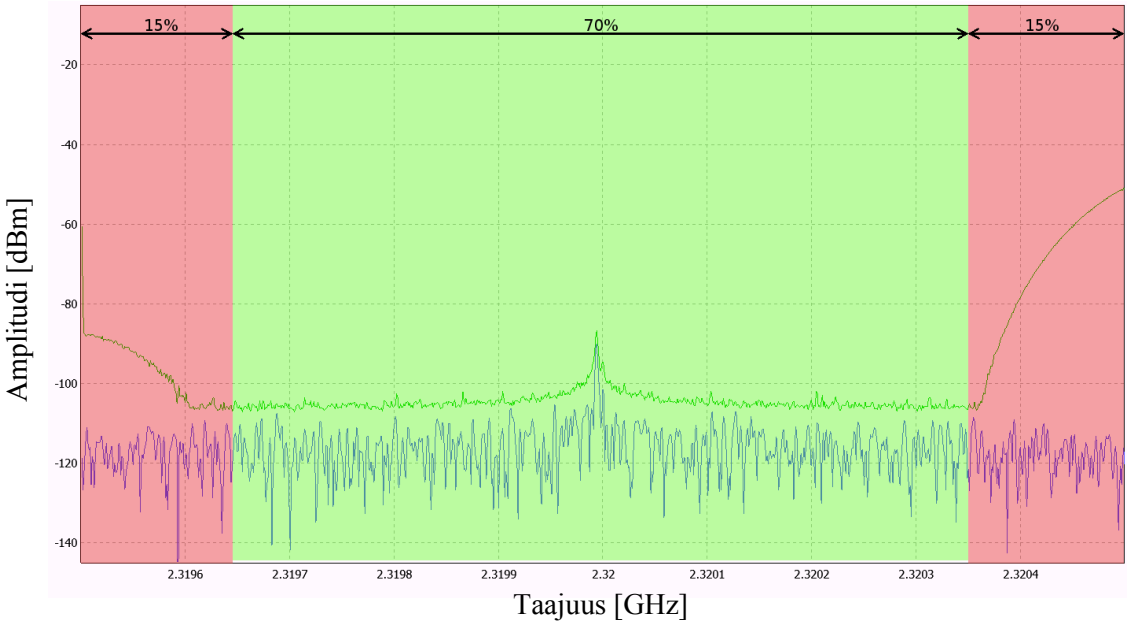
Saavutettavan kalibraation tarkkuus riippuu kalibraatio-ohjelmassa käytetyistä asetuksista, käytetyn signaaligeneraattorin amplitudi- ja taajuustason tarkkuudesta sekä käytetystä kalibrointimenetelmästä. Esimerkiksi laitteen etuasteen vahvistimien vahvistuksen muuttaminen etuasteen virityksen yhteydessä mahdollistaa kalibroinnin vain kyseisen viritysaskeleen kaistanleveyden tarkkuudella. Yksittäisen viritysaskeleen sisäisiä taajuustason virheen muutoksia ei siis voida ottaa suoraan huomioon tämän tyyppistä menetelmää käytettäessä. Saavutettua kalibraatiota voisikin olla mahdollista parantaa suorittamalla signaaleille myös erillinen viritysaskeleen sisäiset amplitudivirheet huomioiva signaalinkäsittelyyn perustuva kalibrointi. Yhden viritysaskeleen sisäisten amplitudivirheiden taso on tosin alle 1 dB:n luokkaa kaikilla käytettävissä olevilla kaistanleveyden arvoilla, kun laitteen etuasteena käytetään SBX-tytärkorttia. Kuvassa 5.9 on esitetty esimerkkimittaus 25 MHz:n viritysaskeleen sisäisestä taajuusvasteesta -30 dBm:n signaalitasolla.



Kuva 5.9: SBX-tytärkortin taajuusvaste 25 MHz:n kaistanleveyden sisällä

5.4.2 USRP N210 SBX-tytärkortin kaistanvalintasuodatin

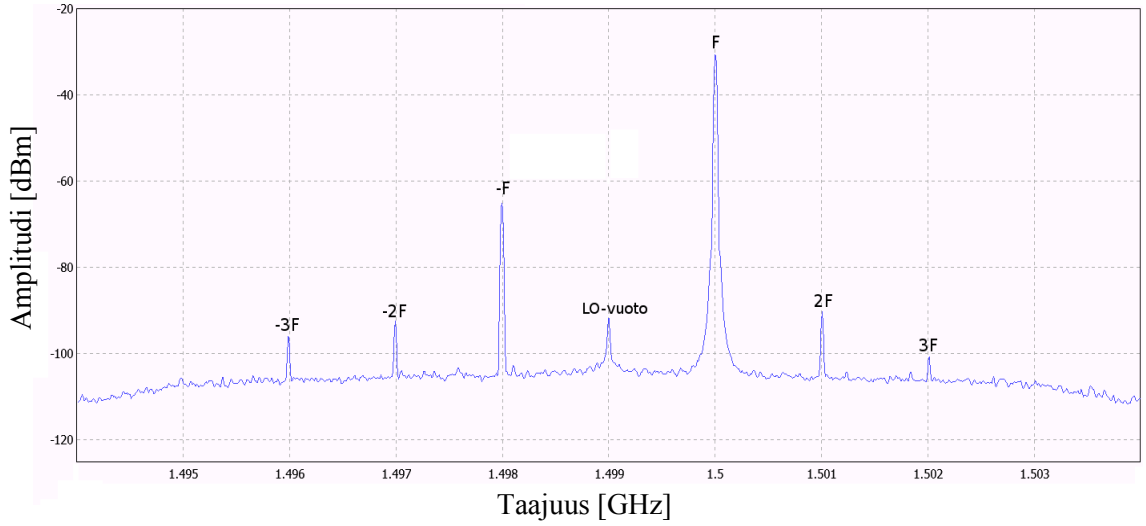
USRP N210:n ADC näytteistää kompleksista I/Q-signaalia 100 MSps nopeudella. Laitteen liitäntärajapintana toimivan Ethernet-liitännän rajoituksista (luku 4.5.4) johtuen hetkellisesti näytteistettävän taajuuskaistan leveys on kuitenkin rajoitettu 25 MHz:n tasolle. Lisäksi käytettävissä olevaa kaistanleveyttä rajoittaa laitteen etuasteena toimivan tytärkortin kaistanvalintasuodattimen rajallinen vaimennus, sillä se mahdollistaa viereisten taajuuskaistojen signaalien sekoittumisen mittauskaistalle [102]. Tilannetta on selvennetty kuvassa 5.10, jossa 2,32 GHz:n keskitaajuudella sijaitsevaa, kaistanleveydeltään 1 MHz:n mittauskaistaa on näytteistetty samalla, kun sen viereinen alemmalla taajuudella oleva kaista on pyyhkäisty käyttäen -40 dBm:n signaalitasoa. Mittauksissa huomattiin, että signaaleja kytkeytyi mittauskaistan reunoille taajuudeltaan jopa kolmen kaistanleveyden etäisyydeltä. Suoritettujen mittausten pohjalta näytteistettävän kaistanleveyden kummaltakin reunalta päätettiinkin hylätä 15% näytteistetyistä FFT-lokeroista. Koko taajuusalueen näytteistys täytyy siis suorittaa käytettyä kaistanleveyttä pienemmissä taajuusaskeleissa.



Kuva 5.10: SBX-tytärkortin näytteistyskaistalle sen viereiseltä alemmalta kaistalta kytkeytyvät signaalit

5.4.3 USRP N210:n näytteistyksen I/Q epätasapaino

Mittauksiin aiheutuu virhettä myös kompleksisessa näytteistyksessä käytettyjen erillisten I/Q-näytteistyskanavien välisestä epätasapainosta, jota aiheuttaa signaaliteiden vahvistusten ero, vaihevirhe sekä LO-vuoto [103]. Kuvassa 5.10 voidaan nähdä kyseisen epätasapainon aiheuttamia ylimääräisiä signaalipiikkejä, kun ohjelmistoradiolla on näytteistetty 1,5 GHz:n taajuudelle generoitua signaalia. Virhesignaalien voimakkuus riippuu vastaanotettujen signaalien voimakkuudesta sekä niiden taajuustason sijainnista suhteessa käytettyyn keskitaajuuteen. Virhesignaalien vaikutusta voidaankin vähentää mittaamalla ja arvioimalla niiden testisignaaliin suhteellista voimakkuutta sekä vaiheeroa ja kompensoimalla mitattuja signaaliarvoja näin saaduilla mittaustuloksilla. Tämä onnistuu esimerkiksi UHD:n sisäänrakennetun ”uhd_cal_rx_iq_balance”-kalibrointityökalun avulla. [101] [104] [105, pp. 49-55]



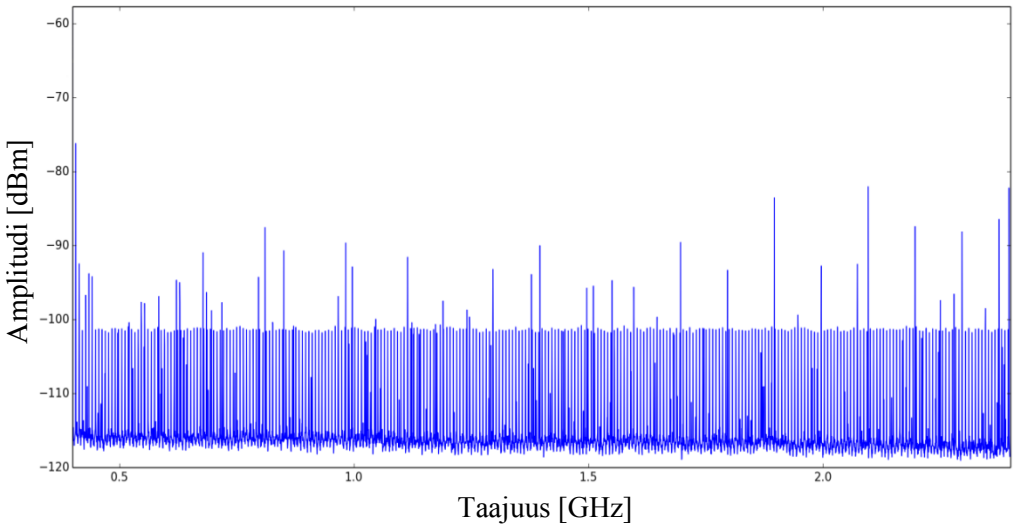
Kuva 5.11: I/Q-epätasapainosta johtuvat ylimääräiset signaalipiikit

Työssä käytetty USRP N210 –ohjelmistoradiosta sekä SBX-tytärkortista koostuva kokoonpano (kuva 4.5) toimii oletusasetuksillaan suoramuunnosvastaanottimena, eli sen LO viritetään suoraan halutulle mitattavalle keskitaajuudelle. Tällöin laitteen LO:n tuottama signaali kytkeytyy suoraan mittauksiin (luku 2.6.2) aiheuttaen mittauskaistan keskitaajuudelle kuvassa 5.11 näkyvän häiriösignaalin. Näytteistettävän kaistanleveyden ollessa haluttua mittausaluetta suurempi, voidaan LO kuitenkin virittää mittausalueen ulkopuolelle. Tällöin näytteistettävästä kaistasta voidaan valita haluttu osa laitteen FPGA:n sisältämien digitaalisten alasmuuntimien avulla, eli laitetta voidaan käyttää matalan välitaajuuden vastaanottimena [102].

Spektrimonitoroinnissa halutaan kuitenkin useimmiten näytteistää kerrallaan mahdollisimman leveä taajuuskaista parhaan aikatason resoluution saavuttamiseksi. SBX-tytärkortilla edellä kuvattua LO:n siirtoa rajoittaa tässä tapauksessa laitteen kaistanvalintasuodattimien vaimennus. LO-vuodon vaikutusta voidaan tässäkin tapauksessa vähentää keskiarvoistuksen avulla tai käyttämällä vieläkin lyhyempää viritysaskelta ja hylkäämällä myös keskitaajuuden ympärillä olevat FFT-lokerot. Myös GNU Radio sisältää funktioita tämän tyyppisen toiminnallisuuden toteuttamiseen. Esimerkiksi DC-offset:sta johtuvaa mittauskaistan keskitaajuudella näkyvää ylimääräistä signaalia voidaan keskiarvoistaa funktion ”usrp_source_sptr.set_auto_dc_offset” avulla. [60]

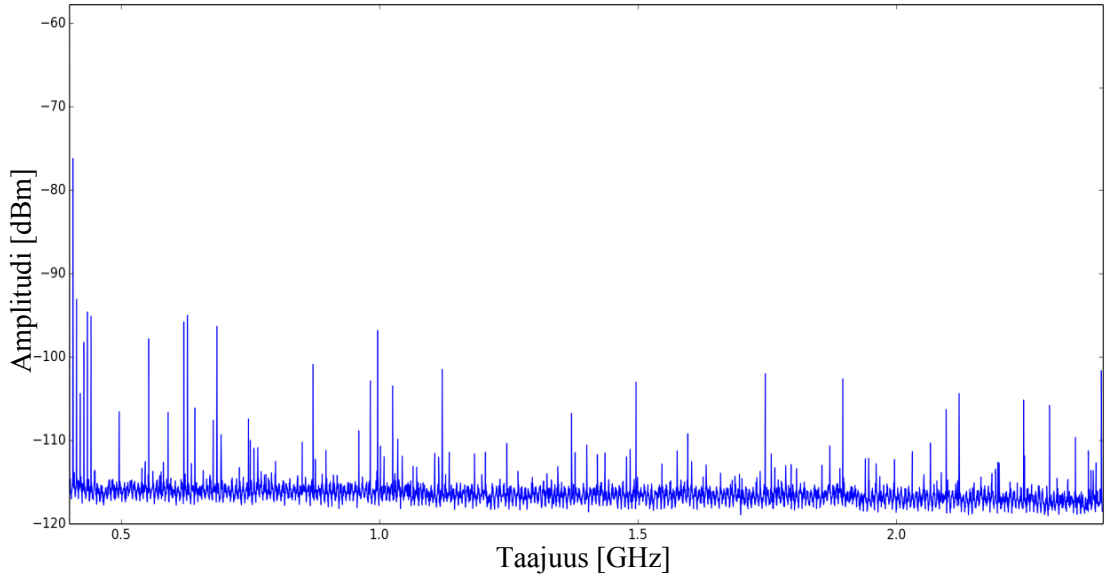
5.4.4 USRP N210:n häiriösignaalit

USRP N210 –ohjelmistoradiojärjestelmä ei itsessään ole valmis mittalaite, eikä se siis automaattisesti huomioi tai kompensoi mittauksiin kytkeytyviä häiriöitä. Järjestelmän häiriösuojaustaso on myös fyysiseltä kokoonpanoltaan varsin puutteellinen ja mittauksiin pääseekin kytkeytymään mitattavan liitännän lisäksi myös järjestelmän itse aiheuttamia sekä sen ulkopuolelta kytkeytyviä signaaleja. Kuvassa 5.12 on esitetty edellä mainittuja häiriöitä selventävä mittaus, jossa USRP N210 –ohjelmistoradiolla ja SBX-tytärkortilla on näytteistetty 400 – 2400 MHz:n taajuusaluetta, kun laitteen sisääntulo on päätetty 50 ohmin päätteellä. Optimitilanteessa kuvassa nähtäisiin pelkästään laitteen kohinapohja. Kuvasta nähdään kuitenkin varsin selvästi, että mittauksiin on kytkeytynyt varsin huomattava määrä erilaisia signaaleja.



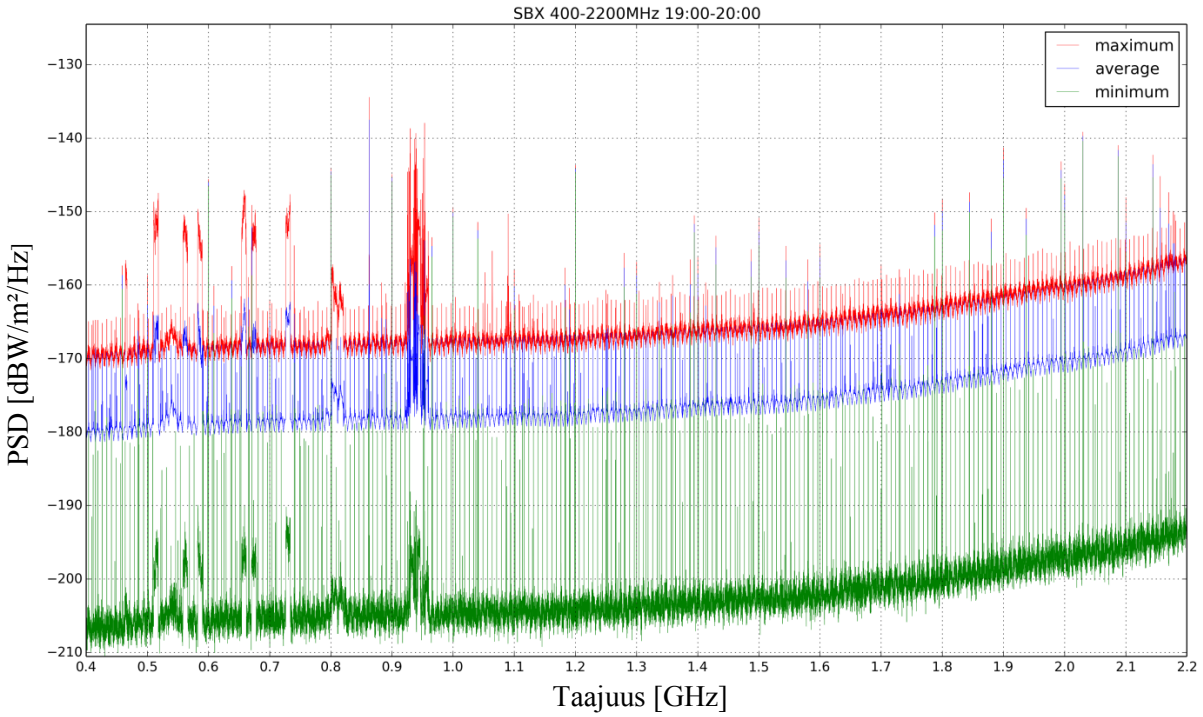
Kuva 5.12: USRP N210 & SBX –kokoonpanolla mitattu, suodattamaton 400 – 2400 MHz:n taajuusalue, kun laitteen sisääntulo on päätetty 50 ohmin päätteellä

Mittauksiin kytkeytyneet laitteen sisäisesti tuottamat signaalit ovat kuitenkin varsin stabiileja. Laitteen sisäisten häiriöiden kompensointiin tuotettiinkin työn yhteydessä ”usrp_zeroes.py” –niminen apuohjelma. Ohjelma tuottaa 50 ohmin päätteen avulla referenssimittauksen, jota voidaan hyödyntää myöhemmin suoritettavien mittausten suodattamisessa. Suodatus perustuu siis kuvassa 5.12 näkyvien ylimääräisten signaalipiikkien amplitudiarvojen vähentämiseen suoritettavasta mittauksesta. Kuvassa 5.13 on esitetty 50 ohmin referenssimittaus, kun edellä kuvattu suodatus on käytössä.

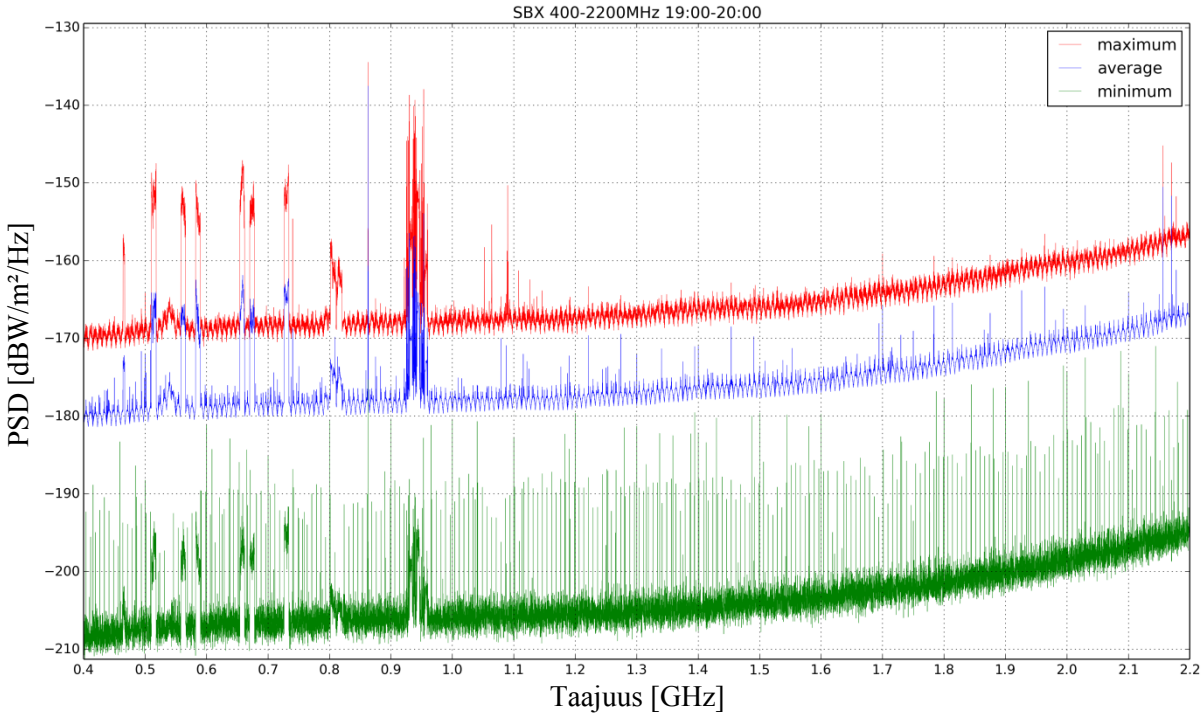


Kuva 5.13: USRP N210 & SBX –kokoonpanolla mitattu, aiemmalla häiriömittauksella kompensoitu 400 – 2400 MHz:n taajuusalue, kun laitteen sisääntulo on päätetty 50 ohmin päätteellä

Toteutettu referenssimittaukseen perustuva suodatus pienentää mittauksissa havaittujen ylimääräisten signaalien lukumäärää varsin huomattavasti. Se ei kuitenkaan sovellu hyvin amplitudiltaan vaihtelevien häiriöiden suodattamiseen. Järjestelmällä mitattujen signaalien suodatusta kehitettiin edelleen työn aikana, jotta myös tämän tyyppisten, laitteen sisäisten häiriöiden suodatus mahdollistettaisiin. Lopulta päädyttiin järjestelmään, jossa mitattuja signaaleja suodatetaan vielä ennen kuvaajien piirtämistä. Signaalien maksimi- ja minimiarvoisia kuvaajia vertailemalla on mahdollista arvioida tietyllä taajuudella näkyvän signaalin alkuperää. Tämä mahdollistaa laitteen sisäisten häiriösignaalien poistamisen mittauksista, 50 ohmin referenssimittauksen avulla toteutetun suodatuksen tapaan. Kyseisen suodatuksen vaikutus on nähtävissä kuvissa 5.14 ja 5.15. Mittaustulosten reaaliaikainen suodatus ja laitteen häiriösuojaustason kehittäminen ovat kehitetyistä suodatusmenetelmistä huolimatta varsin oleellisia laitteen jatkokehityskohteita, sillä ne parantavat järjestelmällä reaaliaikaisesti mitattujen signaalien luotettavuutta.



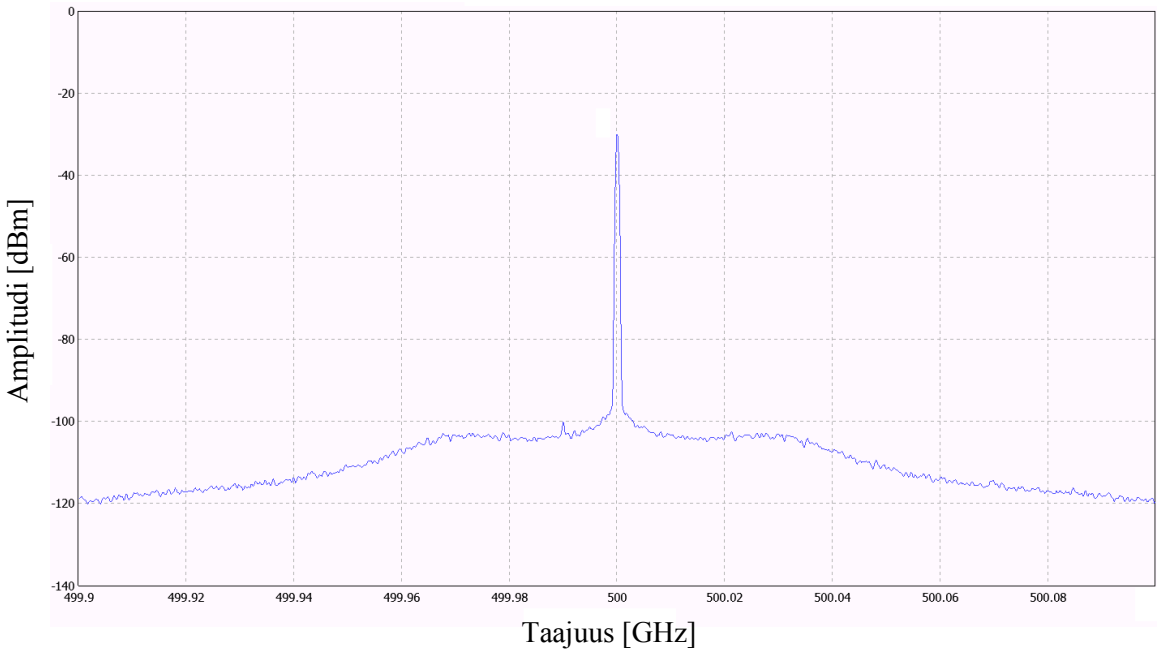
Kuva 5.14: 400 – 2400 MHz välisen taajuusalueen mitattujen signaalien suodattamattomat maksimi-, keski- ja minimiarvoiset tasot tunnin aikajaksolta



Kuva 5.15: 400 – 2400 MHz välisen taajuusalueen mitattujen signaalien suodatetut maksimi-, keski- ja minimiarvoiset tasot tunnin aikajaksolta

5.4.5 USRP N210:n LNA:n epälineaarisuus

Suurilla signaalitasoilla laitteen etuasteena toimivan tytärkortin LNA toimii sen epälineaarisella alueella. Tämä lisää mitattujen signaalien harmonisten komponenttien amplitudia nostaen signaalia ympäröivän taajuusalueen pohjakohinaa (kuva 5.16). Pohjakohinan tason nousu taas vaikeuttaa sen vaikutusalueella olevien muiden pieniamplitudisten signaalien havaitsemista. Pohjakohinan tason nousua voidaan vaimentaa lähinnä luvussa 2.6.6 esitetyillä menetelmillä, eli käyttämällä erillistä ulkoista pienikohinaista vahvistinta ja kytkemällä laitteen etuasteen sisäiset vahvistimet pois käytöstä. Tytärkorttien sisältämät vahvistimet ovat kuitenkin hyödyllisiä, sillä niiden vahvistus on ohjelmallisesti ohjattavissa, mikä mahdollistaa laitteen amplitudivasteen dynaamisen kalibroinnin luvussa 5.4.1 esitetyllä tavalla. [30]



Kuva 5.16: SBX-tytärkortin LNA:n epälineaarisuudesta johtuva pohjakohinan nousu 200 kHz:n kaistanleveydellä ja amplitudiltaan -30 dBm:n tasoisella signaalilla

Luku 6

6 Mittaukset

Tässä diplomityössä pyrittiin tutkimaan ohjelmistoradiojärjestelmän soveltuvuutta radioastronomisten mittausten häiriötekijöiden monitoroinnissa. Lisäksi työssä pyrittiin tuottamaan Ettus Research USRP N210 -ohjelmistoradiolle ohjelmisto, jonka avulla voitaisiin parantaa Metsähovin häiriömonitorointijärjestelmän aikatason resoluutiota sekä herkkyyttä. Tässä luvussa esitellään työssä toteutetulla järjestelmällä suoritettuja mittauksia sekä vertaillaan järjestelmällä saavutettua suorituskykyä Metsähovissa nykyisin häiriömonitoroinnissa käytettävään Agilent FieldFox N9912A –yhdistelmäanalysaattoriin. Lisäksi vertailukohteena käytetään Metsähovissa yleisessä mittauskäytössä olevaa Agilent EXA N9010A-526 –spektrianalysaattoria. Laitteita vertaillaan niiden radioteknisten ominaisuuksien, kuten herkkyyden, taajuustason resoluution, kaistanleveyden selektiivisyyden sekä pyyhkäisynopeuden suhteen.

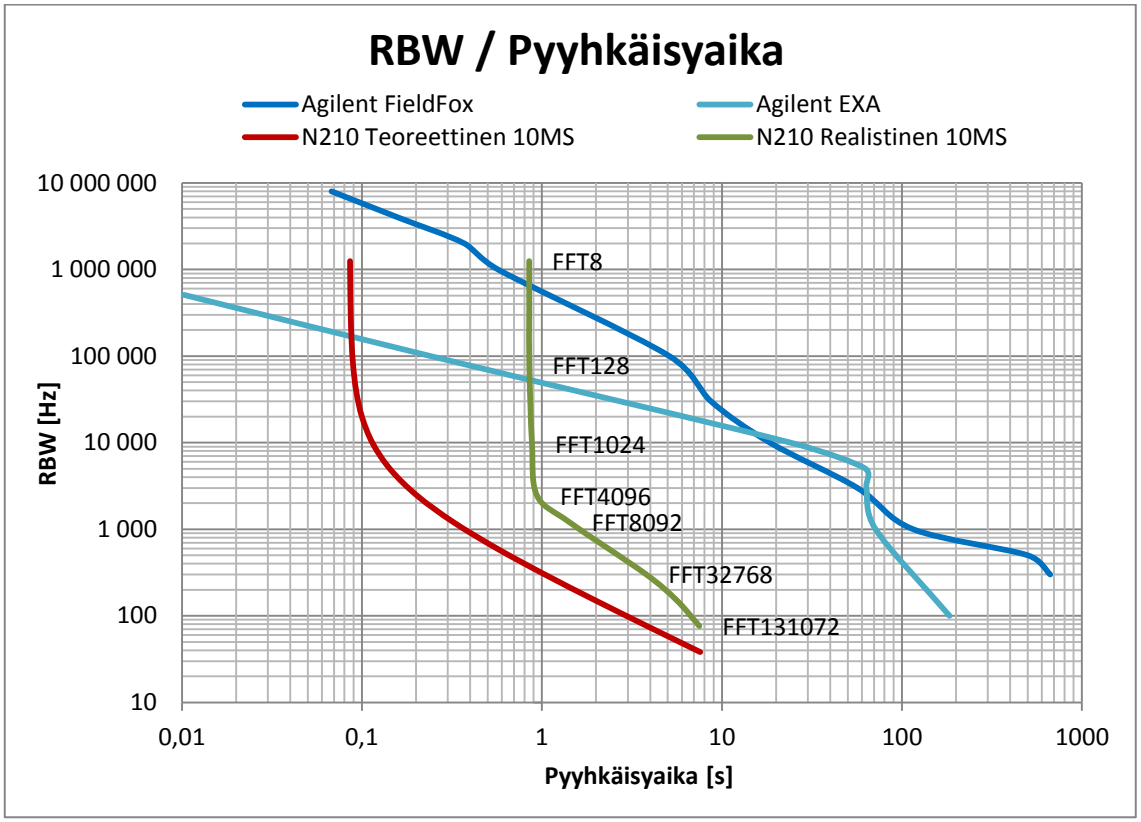
6.1 Järjestelmien aikatason resoluutio

Häiriömonitoroinnissa saavutettava aikatason resoluutio on tämän työn kannalta yksi oleellisimmista järjestelmän ominaisuuksista, sillä kehitetyllä järjestelmällä halutaan vastata dynaamisesti lähetyskaistaansa vaihtavien, vaikeasti ennustettavien häiriölähteiden asettamiin haasteisiin. Ohjelmistoradio mahdollistaa näytteistyskaistan ja käytettyjen asetusten monipuolisen muuntamisen. Tämän ansiosta sitä on mahdollista käyttää esimerkiksi reaaliaikaisena FFT-spektrianalysaattorina, jolloin järjestelmän aikatason resoluutiota rajoittaa vain käytettyyn Fourier'n muunnokseen tarvittavien näytteiden näytteistykseen ja prosessointiin kuluva aika (luku 2.6.4). Laajempaa taajuusaluetta näytteistettäessä ohjelmistoradiota käytetään kuitenkin hybridi-tyyppisen spektrianalysaattorin tapaan, jolloin saavutettavaa aikatason resoluutiota rajoittavat näytteistyksen lisäksi laitteen etupään LO:n asettuminen sekä ohjaussignaalien lähettämiseen ja laitteen sisäisiin toimintoihin liittyvät viiveet.

Kuvassa 6.1 on vertailtu eri järjestelmäkokoonpanoilla saavutettavaa pyyhkäisyn nopeutta mittauksissa käytetyn resoluutiokaistanleveyden funktiona. Mittaukset suoritettiin Metsähovissa häiriömonitoroinnissa nykyisin käytetyllä 400 – 2400 MHz:n taajuusalueella. Kuvasta 6.1 nähdään varsin selvästi, että USRP N210 –ohjelmistoradiolla saavutetaan nykyisin häiriömonitoroinnissa käytössä olevaan Agilent FieldFox:iin verrattuna selvä nopeusetu, kun käytetään alle 600 kHz:n resoluutiokaistanleveyttä. Ohjelmistoradioon perustuvalla järjestelmällä saavutetaan esimerkiksi nykyisin häiriömonitoroinnissa käytettävällä 3 kHz:n RBW:llä noin 90-kertainen pyyhkäisynopeus verrattuna nykyisin käytössä olevaan häiriömonitorointijärjestelmään.

Teoriassa N210 voisi pyyhkäistä mitatun taajuusalueen vielä huomattavasti toteutettua järjestelmää nopeamminkin, sillä laitteen SBX-tytärkortin virityksen asettumisviive on Ettuksen kotisivuilta löytyvien tietojen mukaan vain luokkaa 300 µs. Ongelman muodostavat kuitenkin laitteen ohjaussignaaleihin liittyvät viiveet, jotka rajoittavat laitteistolla saavutettavaa pyyhkäisynopeutta. Esimerkiksi laitteen käyttämän Ethernet-yhtey-

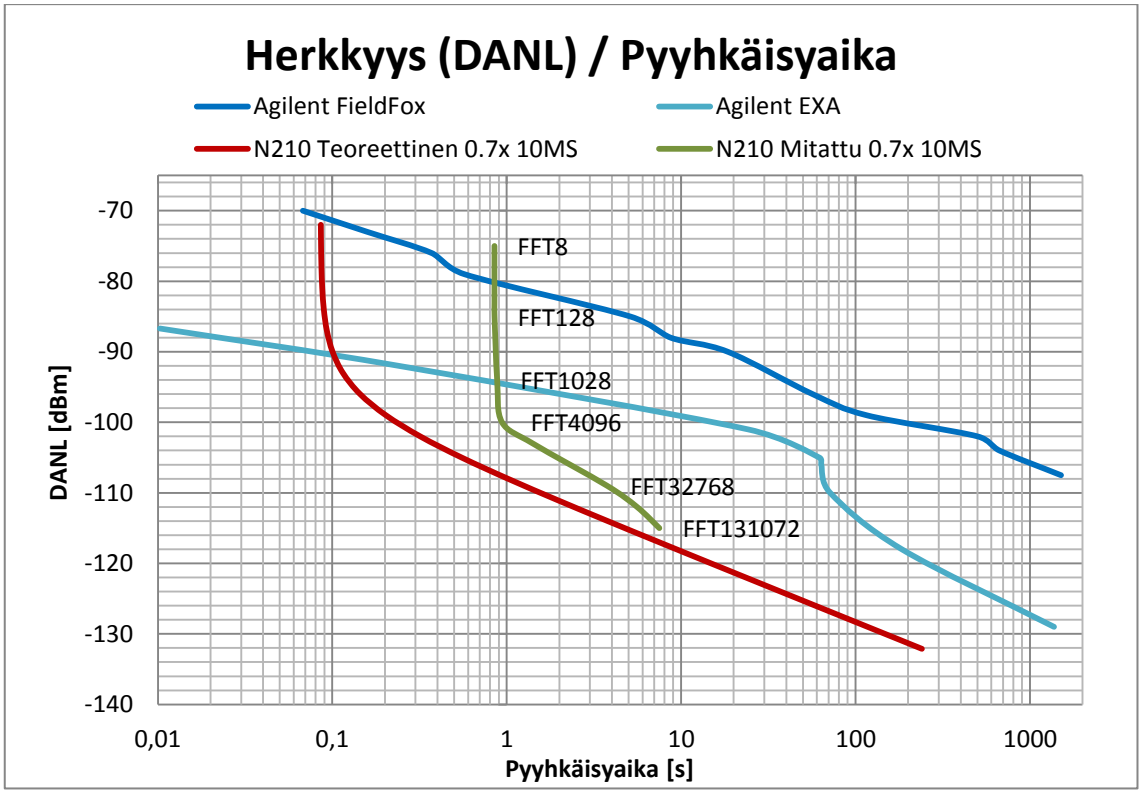
den latenssi aiheuttaa noin millisekunnin lisäviiveen laitteen etupään yksittäiseen viritykseen kuluvaan aikaan. Viivettä aiheuttavat myös laitteen sisäisten FIFO-linjojen täyttämiseen kuluvat ajat [77, p. 61]. Nykyisellä kokoonpanolla järjestelmällä saavutettava pyyhkäisyaika onkin 400 – 2400 MHz:n mittauskaistalla, pienillä FFT:n arvoilla parhaimmillaan noin sekunnin luokkaa. Suuremmilla, yli 4096-alkion FFT-pituuksilla järjestelmän pyyhkäisynopeutta rajoittaa myös Fourier'n muunnosten laskemiseen kuluva aika [35]. Pienillä FFT:n arvoilla voitaisiinkin saavuttaa huomattavia nopeusetuja käyttämällä esimerkiksi FPGA-piiriä laitteen etupään ohjaukseen sekä tarvittavien Fourier'n muunnosten laskemiseen. Esimerkiksi CRUSH-projektissa [77] on ulkoista FPGA-kehitysalustaa käyttämällä saavutettu jopa yli 700-kertainen nopeusetu verrattuna ohjelmalliseen toteutukseen. Kyseinen järjestelmä on tosin näytteistänyt vain tiettyä valittua kaistaa, eli laitetta on käytetty reaaliaikaisena FFT-spektrianalysaattorina. Tässä työssä toteutetun kaltaisella, laajaa taajuusaluetta näytteistävällä järjestelmällä, FPGA:n avulla saavutettava pyyhkäisynopeus olisikin korkeintaan kuvassa 6.1 esitetyn teoreettisen arvon luokkaa. Lisäksi laitteen sisäinen FPGA ei mahdollista kuin 4096-alkion pituisen FFT-algoritmin toteuttamisen, mikä rajoittaa ohuimman sen avulla saavutettavissa olevan RBW:n noin 2,4 kHz:n tasolle.



Kuva 6.1: Järjestelmien 400 – 2400 MHz:n taajuusalueen pyyhkäisyajat RBW:n funktiona

6.2 Järjestelmien herkkyys

Laitteen herkkyyttä, eli sen avulla pienintä havaittavissa olevaa signaalitasoa, voidaan kuvata DANL-arvon avulla. DANL-arvo voidaan määritellä luvussa 2.6.5 esitetyllä tavalla, kun laitteella nähtävissä olevan pohjakohinan taso mitataan laitteen ollessa päätettynä 50 ohmin päätteellä ja valittuna on pienin valittavissa oleva sisääntulovaimennuksen arvo. Kuvassa 6.2 on vertailtu edellä kuvatulla menettelyllä mitattujen, eri järjestelmäkokoonpanojen herkkyyttä, mittauksissa saavutetun pyyhkäisyajan funktiona. Mittauksissa käytettiin pyyhkäistävänä taajuusalueena Metsähovissa nykyisin häiriömonitoroinnissa käytettyä 400 – 2400 MHz:n taajuusaluetta.



Kuva 6.2: Järjestelmien 400 - 2400 MHz taajuusalueen pyyhkäisyssä saavutettu herkkyys suhteessa pyyhkäisyaikaan

Kuvasta 6.2 nähdään, että työssä käytetyllä ohjelmistoradiolaitteistolla saavutetaan muihin käytettävissä oleviin mittauskokoonpanoihin verrattuna parempi herkkyys saavutetun pyyhkäisyajan funktiona. Järjestelmän mittaustulosten luotettavuus vaatii kuitenkin jatkotutkimusten suorittamista. Työssä käytetty USRP N210 esimerkiksi tuottaa varsin runsaasti sisäisiä häiriösignaaleja, jotka rajoittavat mittaustulosten luotettavuutta, kun toimitaan alle -90 dBm:n signaalitasoilla. Lisäksi USRP N210:n sietokyky korkeita signaalitasoja kohtaan on huomattavasti heikompi kuin vertailuissa käytetyillä muilla laitteilla, mikä rajoittaa järjestelmän käytettävissä olevaa dynaamista aluetta ja mahdollisen ulkoisen LNA:n avulla käytettävissä olevaa vahvistustasoa. Esimerkiksi nykyisin häiriömonitoroinnissa käytetty Agilent FieldFox N9912A –yhdistelmäanalysaattori kestää jopa +27 dBm tasoisia sisääntulosignaaleja, kun vastaava USRP N210:n SBX-tytärkortin suositeltu sisääntulosignaalien yläraja on vain -10 dBm:n luokkaa. [63]

6.3 Järjestelmien kaistanleveyden selektiivisyys

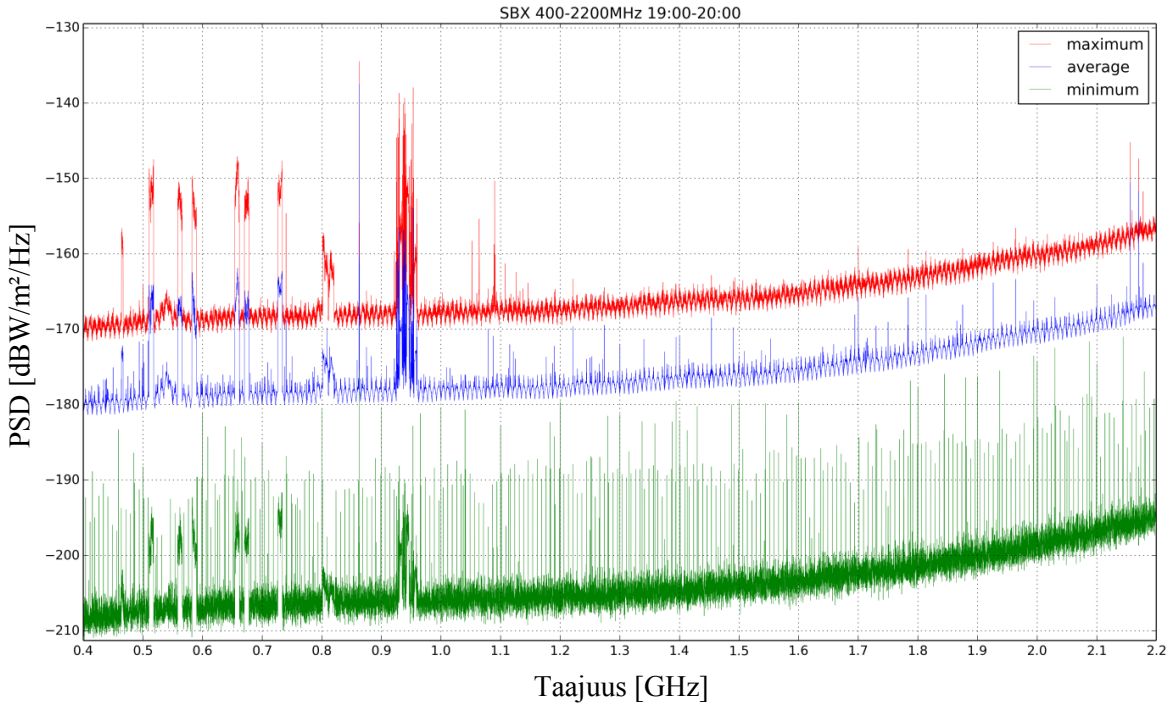
USRP N210 ohjelmistoradiojärjestelmän ja SBX-tytärkortin kaistanleveyden selektiivisyyttä tutkittiin mittaamalla Rohde & Schwarz:n SMB100A signaaligeneraattorilla luotuja testisignaaleja ja vertaamalla mitattujen signaalien -3 dB:n ja -60 dB:n kaistanleveyksiä luvussa 2.6.3 esitetyllä tavalla. Näin mitatut kaistanleveyden selektiivisyyden arvot olivat luokkaa 3,8:1 – 4,3:1, kun mittaukset suoritettiin 400 – 500 MHz:n taajuusalueella. Suuremmilla taajuuksilla mittauksissa esiintyi selkeitä, mittausten ulkopuolisten signaalien aiheuttamia häiriöitä. Häiriöt nostivat mittausten pohjakohinaa, vaikeuttaen mittaustulosten arviointia. Mittaustulosten keskiarvoistaminen olisikin ollut tarpeellista tämän tyyppisten häiriöiden vaikutusten vähentämiseksi. Pienillä taajuuksilla mitattu taajuustason selektiivisyys on kuitenkin varsin vertailukelpoinen nykyään häiriömonitoroinnissa käytetyn Agilent Fieldfox yhdistelmäanalysaattorin luokkaa ~4:1 olevaan kaistanleveyden selektiivisyyteen verrattuna.

Työssä suunniteltiin luvussa 5.4.1 mainitun kalibrointiohjelmiston tavoin toimivan automatisoidun järjestelmän käyttämistä myös tytärkorttien eri parametrien, kuten kaistanleveyden selektiivisyyden ja vahvistimien epälineaarisuuksien tarkempaan tutkimiseen. Järjestelmän alustava kokoonpano myös toteutettiin työn aikana, mutta järjestelmää ei saatu hyödynnettyä mittauksissa, lähinnä SCPI-ohjaukseen liittyvistä liitäntäongelmista johtuen.

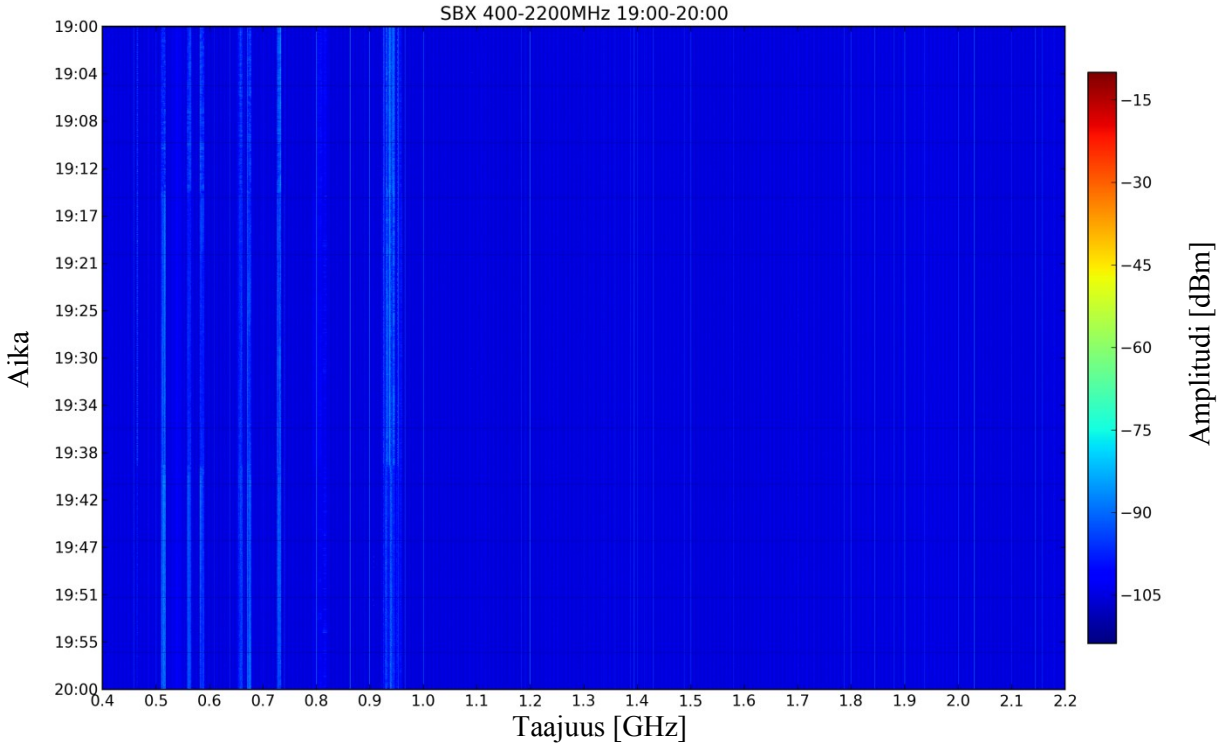
6.4 Järjestelmällä toteutettavissa olevat toimintatilat

Järjestelmä toimii normaalitilassaan seuraavalla tavalla. Haluttu taajuusalue pyyhkäistään ja mitattujen FFT-lokeroiden maksimi-, keski- ja minimiarvot tallennetaan muistiin. Lisäksi löydetyt, tietyn ennalta-asetetun raja-arvon ylittävät signaaliarvot, sekä niiden taajuus ja havaintohetket tallennetaan erilliseen tekstitiedostoon. Mittaustiedot tallennetaan kiintolevylle esimerkiksi tunnin välein tai kerran vuorokaudessa. Mittaustiedoista voidaan tämän jälkeen luoda spektri- ja spektrogrammikuvaajia erillisen ”usrp_imag.py” –ohjelman avulla. Kuvassa 6.3 on esitetty tyypillinen tunnin mittausajalta luotu spektrikuva. Vastaava tunnin mittausajalta luotu spektrogrammikuva on esitetty kuvassa 6.4.

Useampaa ohjelmistoradiolaitetta käytettäessä näytteistettävä taajuusalue voidaan jakaa osiin, mikä parantaa järjestelmällä saavutettavaa pyyhkäisynopeutta entisestään. Esimerkiksi Metsähoviin toteutettu häiriömonitorointijärjestelmä sisältää kaksi Ettus Research N210 ohjelmistoradiolaitetta, mikä mahdollistaa näytteistettävän taajuusalueen, sekä yksittäiseen pyyhkäisyyn kuluvan ajan puolittamisen. Toisessa ohjelmistoradiossa käytetään WBX-tytärkorttia, jonka käytettävissä oleva taajuusalue kattaa 50 – 2200 MHz:n taajuudet. Toisessa ohjelmistoradiossa taas käytetään SBX-tytärkorttia, joka mahdollistaa 400 – 4400 MHz taajuusalueen näytteistämisen. Taajuusalue voidaan siis jakaa esimerkiksi niin, että WBX-tytärkorttia hyödyntävällä ohjelmistoradiolla näytteistetään 100 – 2200 MHz välistä taajuusaluetta ja SBX-tytärkorttia hyödyntävällä ohjelmistoradiolla taas 2200 – 4300 MHz taajuusaluetta. Näin toimivalla järjestelmällä voidaan saavuttaa aiemmin Metsähovissa häiriömonitoroinnissa käytettyyn yhdistelmäanalysaattoriin verrattuna yli satakertainen pyyhkäisynopeus järjestelmän herkkyyden pysyessä samalla tasolla.



Kuva 6.3: 400 – 2400 MHz välisen taajuusalueen mitattujen signaalien maksimi-, keski- ja minimiarvoiset tasot tunnin aikavälillä



Kuva 6.4: 400 – 2400 MHz välisen taajuusalueen signaalitasojen spektrogrammikuvaaja tunnin aikaväliltä

Luku 7

7 Yhteenveto ja tulevaisuuden kehityskohteet

7.1 Yhteenveto

Radiohäiriöitä tuottavien laitteiden ja sovelluskohteiden lukumäärä, sekä niiden hyödyntämien tietoliikenneyhteyksien kaistanleveys kasvaa jatkuvasti, mikä lisää radioastronomisiin mittauksiin kytkeytyviä häiriöitä [10]. Radioastronomisissa mittauksissa käytetyt entistä herkemmät radiovastaanottimet tekevät mittauksista myös entistä herkempiä häiriöitä kohtaan, sillä niiden ansiosta yhä pienemmät häiriötasot voivat aiheuttaa mittauksiin virheitä. Häiriöympäristön monimutkaistuminen sekä mittausten kasvava häiriöherkkyys korostavatkin siis häiriöiltä suojautumisen ja häiriöiden suodattamisen tärkeyttä radioastronomian tieteenalalla. [8]

Häiriöitä tuottavat laitteet kehittyvät jatkuvasti myös entistä kognitiivisempaan suuntaan. Kehittyneet moniantennitekniikat sekä laitteiden radioympäristöstään tietoinen toiminta mahdollistavat entistä pienempiin tehotasoihin perustuvien tietoliikenneyhteyksien toteuttamisen. Toisaalta laitteiden käyttämien taajuuskaistojen ennakointi muuttuu entistä vaikeammaksi, mikä asettaa suuria haasteita myös häiriöitä monitoroivan järjestelmän herkkyydelle sekä aikatason resoluutiolle. [8]

Tässä diplomityössä selvitettiin ohjelmistoradiojärjestelmien soveltuvuutta sekä niiden avulla saavutettavia hyötyjä radioastronomisten mittausten häiriötekijöiden monitoroinnissa. Diplomityö on jatkumoa Jukka-Pekka Porkon vuonna 2010 Metsähoviin tekemälle diplomityölle ”Radio frequency interference in radio astronomy” [1]. Lisäksi työ perustuu Metsähovissa aiemmin tehtyyn häiriömonitorointiin liittyvään tutkimustyöhön sekä häiriöympäristön kartoitukseen [38].

Työssä toteutetulla Ettus Research USRP N210 -ohjelmistoradioon perustuvalla häiriömonitorointijärjestelmällä pyrittiin vastaamaan edellä kuvattuihin haasteisiin. Työn dokumentoinnilla pyrittiin antamaan monipuolinen yleiskuva radioastronomisten mittausten kannalta haitallisista häiriöstä, häiriömonitoroinnista ja ohjelmistoradiojärjestelmistä sekä niiden sisäisestä arkkitehtuurista. Työssä esiteltiin toteutettu häiriömonitorointiohjelmisto, ohjelmiston tuottamiseen käytetyt menetelmät, ohjelmistokirjastot sekä itse toteutetun ohjelmiston toiminta. Työssä myös mitattiin käytetyn ohjelmistoradiolaitteiston radioteknisiä ominaisuuksia, kuten laitteen avulla saavutettavaa pyyhkäisynopeutta, herkkyyttä sekä laitteen kaistanleveyden selektiivisyyttä. Järjestelmää myös vertailtiin sen edellä mainittujen ominaisuuksien osalta Metsähovissa nykyisin häiriömonitoroinnissa käytettävään Agilent FieldFox –yhdistelmäanalysaattoriin. Lisäksi työssä kartoitettiin järjestelmän heikkouksia, millä pyrittiin yksinkertaistamaan järjestelmän jatkokehittämistä.

Työssä toteutettiin USRP N210 –ohjelmistoradiota ohjaava häiriömonitorointiohjelmisto. Ohjelmisto toteutettiin Python-ohjelmointikielellä ja se perustui GNU Radio:n ”usrp_spectrum_sense”-esimerkkiohjelmistoon. Myös muut ohjelmiston osat toteutet-

tiin käyttäen Python-ohjelmointikieltä. Lisäksi ohjelmistossa hyödynnettiin NumPy-ohjelmistokirjastoa.

Työssä toteutetulla järjestelmällä saavutettiin Metsähovin nykyiseen häiriömonitorointijärjestelmään verrattuna lähes satakertainen pyyhkäisynopeus, kun mittaukset suoritettiin nykyisin Metsähovissa häiriömonitoroinnissa käytetyllä 3 kHz:n resoluutiokaistanleveydellä. Toteutetulla järjestelmällä saavutettiin myös suurempi herkkyys nykyistä häiriömonitorointijärjestelmää pienemmillä pyyhkäisyajoilla. Ohjelmistoradioon perustuvan järjestelmän kaistanvalinnan selektiivisyyden arvioitiin myös olevan hyvin vertailukelpoisella tasolla Metsähovin nykyiseen häiriömonitorointijärjestelmään verrattuna. Kaistanvalinnan selektiivisyyteen liittyvissä mittauksissa oli kuitenkin työn toteutuksen aikana tiettyjä ongelmia (luku 6.3), minkä ansiosta kyseisen parametrin kartoitukseen tulisikin tehdä jatkotutkimuksia, jos asia nähdään tarpeelliseksi.

Toteutetulla järjestelmällä on siis mahdollista saavuttaa nykyistä Agilent FieldFox yhdistelmäanalysaattoriin perustuvaa häiriömonitorointijärjestelmää parempi herkkyys ja aikatason resoluutio. Työssä käytetty ohjelmistoradiolaitteisto ei kuitenkaan itsessään ole valmis mittalaite, mikä korostaa laitteen kalibroinnin tärkeyttä. Järjestelmän mittaustulosten luotettavuuden arviointi vaatiikin vielä jatkotutkimuksia. Tästä johtuen järjestelmä nähdäänkin työn kirjoituksen aikaan enemmänkin nykyisen häiriömonitorointijärjestelmän laajennuksena kuin kokonaan sen korvaavana päivityksenä.

7.2 Järjestelmän edut ja jatkokehityskohteet

Ohjelmistoradiojärjestelmän etuina voidaan nähdä sen etäohjattavuus, joustavuus sekä avoin, modulaarinen laiterakenne, joka mahdollistaa järjestelmän monipuolisen laajentamisen. Ohjelmistoradiota voidaan myös käyttää yleisesti kehitys- ja testityökaluna, sillä se mahdollistaa erilaisten vastaanottomenetelmien; adaptiivisten-, spatiaalisten- ja tilastollisten suodatusmenetelmien, sekä erilaisten järjestelmäkokoonpanojen monipuolisen testaamisen. Luotettavien absoluuttisten mittausten suorittaminen USRP N210 – ohjelmistoradiojärjestelmällä vaatii kuitenkin laitteen kalibrointia sekä laitteen ulkoisten- ja sisäisten virhelähteiden huomiointia. Lisäksi laitteen häiriösuojauksen- sekä suursignaalien siedon taso on varsin heikko verrattuna kaupallisiin mittalaitteisiin. Laitteen edellä kuvatut heikkoudet nähdäänkin sen oleellisimpina jatkokehityskohteina.

Toteutettu järjestelmä on lisäksi varsin herkkä erilaisia Ethernet-yhteyden virhetiloja kohtaan. Jo yhtä USRP N210:tä käytettäessä laitteiston mittaamissa signaaleissa on nähtävissä usein huomattavia virheitä, jos laitteistoa ohjaavalla tietokoneella suoritetaan samanaikaisesti muita verkkoyhteyksiä hyödyntäviä tehtäviä. Järjestelmä on kuitenkin toiminut Metsähovin palvelimeen asennettuna varsin vakaasti, eikä edellä mainittuja virheitä ole mittauksissa juurikaan esiintynyt. Järjestelmän virheherkkyyden tutkiminen sekä mahdollisten virheenkorjausmenetelmien toteuttaminen nähdään kuitenkin varsin oleellisena järjestelmän jatkokehityskohteena. Virheettömien pyyhkäisyjen tärkeyttä korostaa se, että mittaustulokset tallennetaan jatkossa todennäköisesti vain vuorokauden välein, jolloin yksittäinen virheellinen pyyhkäisy voi aiheuttaa huomattavia virheitä koko vuorokauden aikana mitattuihin maksimiarvoisiin signaaliarvoihin.

Toteutettu ohjelmisto mahdollistaa myös aiemmin käytössä olleen käännettävän RFI-antennin ohjaamisen. Antennia ohjaamalla mittauksia voidaan suorittaa nykyisen häiriömonitorointijärjestelmän tavoin neljästä pääilmansuunnasta, mikä yksinkertaistaa häiriölähteiden tunnistamista ja mahdollistaa häiriöympäristön tarkemman kartoituksen. Nykyisin antennin ohjaus suoritetaan kuitenkin Metsähovin aiemman häiriömonitorointiohjelmiston kautta. Järjestely ei ole erityisen yhteensopiva tässä työssä toteutetun järjestelmän kanssa, sillä antennin suuntatietoja joudutaan kysymään yksittäisten nopeiden pyyhkäisyjen välissä, mikä hidastaa järjestelmällä saavutettavaa pyyhkäisynopeutta. Tästä syystä myös häiriöantennin ohjauksen jatkokehittäminen nähdään oleellisena osana koko järjestelmän toiminnan kehittämisessä.

Työssä toteutettu ohjelmisto on itsessään varsin yksinkertainen, eikä se hyödynnä kaikkia ohjelmistoradiojärjestelmän tarjoamia etuja. Esimerkiksi järjestelmän FPGA-piirin käyttö itse signaalinkäsittelyssä voisi mahdollistaa varsin huomattavien nopeusetujen saavuttamisen varsinkin silloin, kun laitetta käytetään reaaliaikaisena FFT-spektrianalysaattorina. Lisäksi se voisi mahdollistaa ohjelmistoradiojärjestelmän suorituskyvyn pullonkaulana toimivan Ethernet-yhteyden asettamien rajoitusten välttämisen. Järjestelmän FPGA-ajurin monimutkainen rakenne vaikeuttaa kuitenkin laitteen sisäisen FPGA:n jatkokehittämistä. Mahdolliseen FPGA-jatkokehitykseen olisikin kannattavaa käyttää ulkoista FPGA-kehitysalustaa esimerkiksi CRUSH-projektissa esitetyllä tavalla [77]. Tämän tyyppinen järjestely mahdollistaisi signaalinkäsittelyssä käytetyn FPGA-kehitysalustan päivittämisen erillään järjestelmän näytteistyksestä huolehtivista osista, parantaen järjestelmän modulaarisuutta entisestään. USRP:n oletusajurien pääpiirteisen toiminnallisuuden säilyttäminen mahdollistaisi lisäksi laitteen käyttämisen rinnakkaisena testialustana ulkoisella FPGA:lla kehitettyjen algoritmien ja toiminnallisuuksien testaamisessa. Ulkoisen FPGA-kehitysalustan ja USRP:n rajapintana olisi mahdollista käyttää esimerkiksi laitteen MIMO-liitäntää [77] tai laitteen Ethernet-liitäntää lähteessä [99, pp. 70-73] esitetyllä tavalla. MIMO-liitäntään liittyvät reilusti pienemmät viiveet sekä liitännän suurempi kaistanleveys tekevät siitä kuitenkin oletusarvoisesti Ethernet-liitäntää käytännöllisemmän valinnan.

Laitteen etupään virittämiseen liittyvät viiveet ovat eräs toteutetulla järjestelmällä sekä yleisesti nykyisillä spektrianalysaattoreilla saavutettavissa olevaa pyyhkäisynopeutta eniten rajoittavista tekijöistä. Laitteiden prosessointitehon kasvaessa voisikin olla mahdollista kehittää laskennallisia menetelmiä, joilla korkeammilla taajuuksilla sijaitsevia signaaleja ei havaittaisi nykyisin yleisesti radiovastaanottimissa käytetyn heterodyne-rakenteen avulla, vaan suoraan laskennallisesti erillisten, eri nopeuksilla näytteistävien A/D-muuntimien avulla saatuja mittaustuloksia vertailemalla. Esimerkiksi lähteessä [106] on saatu kehitettyä näin järjestelmä, jossa kolmea 50 MSps nopeudella näytteistävää USRP –ohjelmistoradiolaitteistoa hyödyntäen on pystytty havaitsemaan signaaleja reaaliaikaisesti jopa 900 MHz:n hetkelliseltä kaistanleveydeltä. Tämän kaltaisten laskennallisten menetelmien tutkiminen onkin erittäin mielenkiintoinen jatkokehityskohde myös tässä työssä kehitettyä järjestelmää ajatellen.

Ohjelmistoradion eräs suurimmista käyttökohteista on uusiin entistä kognitiivisempiin radiojärjestelmiin liittyvä tutkimustyö. Suuri osa kognitiivisiin radiojärjestelmiin liittyvästä tutkimuksesta keskittyy käytettävällä taajuusalueella olevien signaalilähteiden havaitsemiseen. Tässä työssä toteutetun häiriömonitorointijärjestelmän kannalta oleellista on se, että tämän tyyppisissä tutkimuksissa esitetyt ongelmat ja ratkaisut ovat usein

yleistettävissä kattamaan minkä tahansa signaalin havaitsemisen. Tämä mahdollistaa siis kyseisten aihepiirien tutkimuksissa saatujen tulosten hyödyntämisen myös häiriömonitoroinnissa. Lisäksi kognitiivisten radiojärjestelmien yleistyessä on hyvin todennäköistä, että järjestelmiin kehitetään toiminnallisuudeltaan spektrin havaitsemiseen keskittyviä nopeita ASIC-piirejä, joita voi olla mahdollista hyödyntää myös itse häiriömonitorointijärjestelmän jatkokehittämisessä. [77]

Tulevaisuuden kognitiiviset radiojärjestelmät tulevat myös olemaan entistä tietoisempia omasta radioympäristöstään sekä kyseisen ympäristön häiriölähteistä. Lisäksi järjestelmät tulevat jakamaan oppimaansa tietoa keskenään. Järjestelmien yleistyessä olisikin hyödyllistä, jos laitteiden itsensä mittaamaa tietoa voitaisiin hyödyntää häiriöympäristöön liittyvien tietokantojen kehityksessä, sekä itse ympäristön häiriöiden monitoroinnissa. Tämän kaltaisen älykkään järjestelmän hyödyntäminen voisi parantaa häiriöiden ennustettavuutta ja edesauttaa radioastronomisten mittausten häiriötöntä suorittamista sekä erilaisten radioastronomiassakin käytettävien häiriönsuodatusmenetelmien kehittämistä. [11]

Viitteet

[1] J.-P. G. Porko, "Radio frequency interference in radio astronomy", Diplomityö, Espoo: Teknillinen korkeakoulu (TKK), Elektroniikan, tietoliikenteen ja automaation tiedekunta, 2011.

[2] Ursa, ”Radioastronomian historia”, Tähtitieteellinen yhdistys Ursa ry. Verkkosivu. Haettu 29.12.2014 osoitteesta https://www.ursa.fi/wiki/Radioastronomia/Historia.

[3] Committee on Radio Astronomy Frequencies (CRAF), CRAF Handbook for Radio Astronomy (3th edition), Strasbourg: European Science Foundation, 2005.

[4] Metsähovi, ”Metsähovin tutkimusprojektit”, Aalto-yliopisto. Verkkosivu. Haettu 29.12.2014 osoitteesta http://metsahovi.aalto.fi/en/research/projects/.

[5] S. Chaudhari, Spectrum Sensing for Cognitive Radios: Algorithms, Performance and Limitations, Väitöskirja, Espoo: Aalto-yliopisto, Sähkötekniikan korkeakoulu, 2012.

[6] Viestintävirasto, ”Taajuusjakotaulukko”. Verkkosivu. Haettu 29.12.2014 osoitteesta https://www.viestintavirasto.fi/attachments/Taajuusjakotaulukko_31122013.pdf.

[7] E. Häkkinen, K. Fallström, A. Haapalinna ja P. Kärhä, ”Häiriökysymykset - Häiriöt mittauksissa”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://metrology.hut.fi/courses/s108-180/hairmit.pdf.

[8] W. A. Baan, ”RFI Mitigation in Radio Astronomy (RFI2010 - RFI Mitigation Meeting)”, 2010. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.ursi.org/proceedings/procGA11/ursi/J08-3.pdf.

[9] P. C. H. L. A. Crane, ”Estimating Harmful Levels of Radio-Frequency Radiation”, *Light Pollution, Radio Interference and Space Debris, Astronomical Society of Pasific Conference Series,* osa/vuosik. 17, nro 112, pp. 258-266, 1990.

[10] Huawei, ”White paper on spectrum,” Helmikuu 2013. Verkkosivu. Haettu 29.12.2014 osoitteesta www.huawei.com/ilink/en/download/HW_204545.

[11] J. Lundén, Spectrum Sensing for Cognitive Radio and Radar Systems, Väitöskirja, Espoo: Teknillinen korkeakoulu (TKK), Elektroniikan, tietoliikenteen ja automaation tiedekunta, 2009.

[12] IEEE, ”The End of Spectrum Scarcity”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://spectrum.ieee.org/telecom/wireless/the-end-of-spectrum-scarcity.

[13] S. Bergen, "Rinnakkaisprosessointi Kognitiivisessa Radiossa", Kandidaatintyö, Espoo: Teknillinen korkeakoulu, tietoliikenteen ja automaation tiedekunta, 2009.

[14] K. G. Jansky, ”Electrical Disturbances Apparently of Extraterrestrial Origin,” 1933 (1998 uusintajulkaisu). Verkkosivu. Haettu 29.12.2014 osoitteesta https://www.ieee.org/documents/jansky.pdf.

[15] S. Pohjolainen ja K. Wiik, ”Tuorlan observatorio - XFYS4336 Havaitseva tähtitiede II,” Kevät 2014. Verkkosivu. Haettu 29.12.2014 osoitteesta http://tube.utu.fi/courses/lib/exe/fetch.php?media=luento1.pdf.

[16] A. Lähteenmäki ja M. Tornikoski, ”Metsähovin aineistopankki - Radioastronomian perusteita”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.jokioinen.fi/fi/Aineistopankki/Radioastronomia.pdf.

[17] National Radio Astronomy Observatory (NRAO), ”Grote Reber biography”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.nrao.edu/whatisra/hist_reber.shtml.

[18] Turun yliopisto, ”Tähtitieteen peruskurssi I - luentomateriaali - Luku 18”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.astro.utu.fi/edu/kurssit/ttpk1/ttpkI/18Maailmankaikkeus.html.

[19] R. Wielebinski ja T. Wilson, ”The Early History of European Radio Astronomy, Astronomische Nachrichten 328(5), 375–446.,” astronomicalheritage.net, 2007. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www2.astronomicalheritage.net/index.php/show-theme?idtheme=18.

[20] S. Gulyaev ja P. Banks, ”Radio sky and the right to observe it”, *Third International Starlight Conference*, Lake Tekapo, New Zealand, 2012.

[21] Viestintävirasto, ”Radiotaajuuksien käyttö”. Verkkosivu. Haettu 04.03.2014 osoitteesta https://www.viestintavirasto.fi/taajuudet/radiotaajuuksienkaytto.html.

[22] Metsähovi, ”Kvasaaritutkimuksen peruskäsitteitä”, Aalto-yliopisto. Verkkosivu. Haettu 29.12.2014 osoitteesta http://metsahovi.aalto.fi/fi/research/projects/quasar/introduction/concept/.

[23] JPL, ”Basics of Radio Astronomy for the Goldstone-Apple Valley Radio Telescope”, 1998. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www2.jpl.nasa.gov/radioastronomy/radioastronomy_all.pdf.

[24] Aalto-yliopisto, ”S-92.145 Radioastronomia -kurssin luentomateriaali”, 2006. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.metsahovi.fi/edu/radast2006/luennot.html.

[25] A. Enqvist, Radiotaajuiset häiriöt ja niiden tunnistaminen SMOS-ympäristösatelliitin mittauksista Suomen alueella, Diplomityö, Espoo: Aalto-yliopisto Sähkötekniikan korkeakoulu, 2012.

[26] R. P. Millenaar ja H. J. Stiepel, ”CRAF - On Self-generated RFI at Radio Astronomy Sites,” 2003. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.craf.eu/craf0401.pdf.

[27] International Telecommunication Union (ITU), ”Recommendation ITU-R RA.769-2 - Protection criteria used for radio astronomical measurements”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.itu.int/dms_pubrec/itu-r/rec/ra/R-REC-RA.769-2-200305-I!!PDF-E.pdf.

[28] A. M. Niknejad, ”University of California, Berkeley - EECS 142 -kurssin luentokalvot (luento 9)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://rfic.eecs.berkeley.edu/~niknejad/ee142_fa05lects/pdf/lect9.pdf.

[29] H. Karttunen, Ursa ja T. observatorio, ”Zubenelgenubi (tähtitieteen tietokanta)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.astro.utu.fi/zubi/astro.htm.

[30] Agilent Technologies, ”Spectrum Analysis Basics (Application Note 150)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://cp.literature.agilent.com/litweb/pdf/5952-0292.pdf.

[31] Teknillinen Korkeakoulu - MIKES TKK, ”S-108.1010 Mittaustekniikan perus-

teet A -kurssin materiaali”, 2006. Verkkosivu. Haettu 29.12.2014 osoitteesta http://metrology.tkk.fi/courses/S-108.1010/index.php.

[32] C. Rauscher, ”Fundamentals of Spectrum Analysis (Rohde & Schwarz)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://mwrf.com/site-files/mwrf.com/files/uploads/2013/09/SpecAnFundamentals.complete_20130903135041_710227.pdf.

[33] Rohde & Schwarz, ”Implementation of Real-Time Spectrum Analysis (White Paper)”. Verkkosivu. Haettu 29.12.2014 osoitteesta www.rohde-schwarz.de/file/1EF77_2e.pdf.

[34] P. Denisowski, ”Spectrum Analyzers vs. Monitoring Receivers (Rohde & Schwarz)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.denisowski.org/Articles/Denisowski%20%20Spectrum%20Anlyzers%20vs.%20Monitoring%20Receivers.pdf.

[35] Agilent Technologies, ”Performance Spectrum Analyzer Series (Sovellusohje)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://cp.literature.agilent.com/litweb/pdf/5980-3081EN.pdf.

[36] R. Overdorf ja R. Bordow, ”Understanding Probability of Intercept for Intermittent Signals (Agilent Technologies)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.keysight.com/upload/cmc_upload/All/21March2013Slides.pdf.

[37] M. Uunila, "Improving geodetic VLBI: UT1 accuracy, latency of results, and data quality monitoring", Väitöskirja, Espoo: Aalto-yliopisto, 2013.

[38] P. Kirves, J. Kallunki ja J. Wagner, ”RFI in Metsähovi - measurements and effects (RFI2010 - RFI Mitigation Meeting)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.astron.nl/rfi/presentations/RFI2010.session.8.1.Kirves.pdf.

[39] Metsähovi, ”Metsähovin radiotutkimusasema”, Aalto-yliopisto. Verkkosivu. Haettu 29.12.2014 osoitteesta http://metsahovi.aalto.fi/en/about/.

[40] Metsähovin radio-observatorio, ”Lehdistökuvat,” Aalto-yliopisto. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.metsahovi.fi/lehdistokuvat/.

[41] M. Tornikoski, H. B ja M. Uunila, ”Metsähovi Radio Observatory - Annual Report,” Aalto University, 2011.

[42] P. Kirves, J. Kallunki ja J. Wagner, ”RFI in Metsähovi Radio Observatory - measurements and effects (RFI mitigation workshop - RFI2010)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://pos.sissa.it/archive/conferences/107/027/RFI2010_027.pdf.

[43] M. Tornikoski, A. Mujunen, B. Holmberg ja M. Uunila, ”Metsähovi Radio Observatory Annual Report 2010,” Aalto-yliopisto, 2010.

[44] Keysight Technologies, ”FieldFox -yhdistelmäanalysaattorien ohjelmointiohje”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://na.support.keysight.com/fieldfox/help/Programming/webhelp/FFProgrammingHelp.htm.

[45] Agilent Technologies, ”Agilent N9912 Fieldfox tekninen yleiskatsaus”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://cp.literature.agilent.com/litweb/pdf/5989-8618EN.pdf.

[46] L. Zhou, USRP2-based Software Defined Radar Receiver, Diplomityö, Espoo: Aalto-yliopisto, 2011.

[47] J. I. Mitola, ”Software radios-survey, critical evaluation and future directions”,

Telesystems Conference, 1992.

[48] A. F. Y. Youssef, K. M. H.-A. Hassan, M. G. Mostafa ja M. T. Saad, ”Implementation of a wireless OFDM system using USRP 2 and USRP N210 kits”, Master's thesis, Faculty of Engineering, Cairo University, Giza, Egypt, 2012.

[49] J. Keisala, "Ohjelmistoradio - Katsaus käyttötarkoituksiin, toimintaperiaatteisiin, laitteistoihin sekä hyötyihin", Tutkintotyö, Tampere: Tampereen ammattikorkeakoulu, 2010.

[50] T. Hakkarainen, "Ohjelmistoradion hyödyntäminen tiedonsiirron tutkimisessa LVDC-verkossa", Diplomityö, Lappeenranta: Lappeenrannan teknillinen yliopisto, Energia- ja sähkötekniikan osasto, 2010.

[51] Y. N. Papantonopoulos, ”EETimes - High-speed ADC technology paves the way for software defined radio”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.eetimes.com/document.asp?doc_id=1272384.

[52] The Wireless Innovation Forum, ”Software Defined Radio”. Verkkosivu. Haettu 01.12.2014 osoitteesta http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf.

[53] L. M. d. M. Antunes, "Software Defined Radio em FPGA", Santiago, Portugali: Aveiron yliopisto, elektroniikan, telekommunikaation ja informaatiotekniikan laitos, 2009.

[54] Wipro Technologies, ”Software-Defined Radio, White Paper, A Technology Overview”, Huhtikuu 2002. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.invictusnetworks.com/faq/RF%20Technical%20Info%20and%20FCC%20Regs/Software%20Defined%20Radio%20Whitepaper.pdf.

[55] T. Helmijoki, "Ohjelmistoradio, USRP-kehitysalusta", Tutkintotyö, Tampere: Tampereen Ammattikorkeakoulu, 2008.

[56] L. C. d. O. Matos, Design Platform for Software Defined Radio Systems, Master's Thesis, Santiago, Portugali: Aveiron yliopisto, elektroniikan, telekommunikaation ja informaatiotekniikan laitos, 2012.

[57] M. Rauhanummi, ”Integration of agile RF front-end to FPGA development board”, Diplomityö, Oulun yliopisto - tietotekniikan osasto, 2013.

[58] SDRForum, ”www.sdrforum.org - Cognitive Radio Definitions and Nomenclature,” 2008. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-P-0009-V1_0_0_CRWG_Defs.pdf.

[59] W. Wang, ”Spectrum Sensing for Cognitive Radio”, *Intelligent Information Technology Application Workshops, 2009. IITAW '09 Third International Symposium on Intelligent Information Technology Application Workshops*, Nanchang, 2009.

[60] J. Ollikainen, "Laajakaistaisen RF-etupään suunnittelu kognitiiviseen radiovastaanottimeen", Diplomityö, Espoo: Aalto-yliopiston teknillinen korkeakoulu, 2010.

[61] M. Marjamäki, "Kognitiivinen radio - Vapaat taajuusspektrialueet nyt ja tulevaisuudessa", Tutkintotyö, Lahti: Lahden Ammattikorkeakoulu, tietotekniikan koulutusohjelma, 2012.

[62] Ettus Research LLC, ”USRP-kehitysalustojen ja tytärkorttien kytkentäkaaviot”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://files.ettus.com/schematics/.

[63] Ettus Research LLC, ”Ettus Research:n kotisivut”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.ettus.com/.

[64] K. Dabčević, "Evaluation of Software Defined Radio Platform with respect to implementation of 802.15.4 ZigBee", Diplomityö, Västerås: Mälarinlaakson korkeakoulu (Mälardalens högskola), 2011.

[65] S. Scott, "RHINO - Reconfigurable Hardware Interface for computatioN and radiO", Master's thesis, University of Cape Town, 2011.

[66] Ettus Research LLC, ”USRP N210 Datalehti”. Verkkosivu. Haettu 29.12.2014 osoitteesta https://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf.

[67] C. Dick, ”EETimes - A case for using FPGAs in SDR PHY”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.eetimes.com/document.asp?doc_id=1277752.

[68] R. H. L. Stroop, ”Enhancing GNU Radio for Run-Time Assembly of FPGA-Based Accelerators”, Master's thesis, Virginia Polytechnic Institute and State University, 2012.

[69] GNU Radio, ”GNU Radio -projektin kotisivu”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.org/.

[70] Ettus Research LLC, ”Ohje RF-tytärkortin valintaan”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.ettus.com/content/files/kb/Selecting_an_RF_Daughterboard.pdf.

[71] Ettus Research LLC, ”Ettus Research USRP-ohjelmistoradioiden tiedot”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://files.ettus.com/.

[72] A. Martian, ”Real-time Spectrum Sensor based on USRP”, *10th International Conference on Communications (COMM)*, Bucharest, Romania, 2014.

[73] N. Manicka, "GNU Radio testbed", Master's Thesis, University of Delaware, 2007.

[74] Epic Solutions, ”Bitshark USRP BURX kortin tuotesivu”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://epiqsolutions.com/burx.php.

[75] Ettus Research LLC, ”USRP-laitteiden kaistanleveys”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.ettus.com/kb/detail/usrp-bandwidth.

[76] Ettus Research LLC, ”Sovellusohje USRP laitteiden synkronisointiin ja MIMO-yhteyksien käyttöön”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.ettus.com/content/files/kb/mimo_and_sync_with_usrp_updated.pdf.

[77] G. F. I. Eichinger, ”CRUSH: Cognitive Radio Universal Software Hardware", Master's Thesis, Northeastern University, Boston, Massachusetts, 2012.

[78] GNU (Free Software Foundation), ”GNU GPL lisenssi”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.gnu.org/licenses/gpl.html.

[79] Aalto-yliopisto (S. Vilmusenaho), ”Aalto-yliopiston Wiki - Avoimen lähdekoodin lisenssit”. Verkkosivu. Haettu 29.12.2014 osoitteesta https://wiki.aalto.fi/pages/viewpage.action?pageId=56199997.

[80] S. Mahmood, ”Software Defined Radio for Wireless Sensor & Cognitive Networks", Tutkintotyö, Vaasa: Vaasan Ammattikorkeakoulu, Informaatioteknologian laitos, 2011.

[81] GNU Radio, ”"Suggested Reading" -linkit”. Verkkosivu. Haettu 29.12.2014 osoitteesta

http://gnuradio.org/redmine/projects/gnuradio/wiki/SuggestedReading.

[82] T. Rondeau, ”Exposing GNU Radio - Developing and Debugging (GNU Radio Tutorial)”, 2012. Verkkosivu. Haettu 04.06.2014 osoitteesta http://www.trondeau.com/storage/tutorial/gr_tutorial.pdf.

[83] Y. Amn, M. Karim, M. Gamal ja M. Taha, "Implementation of a wireless OFDM system using USRP 2 and USRP N210 kits", Master's thesis, Giza, Egypt: Faculty of Engineering, Cairo University, 2012.

[84] GNU Radio, ”GNU Radio - VOLK”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk.

[85] T. W. Rondeau, N. McCarthy ja T. O’Shea, ”SIMD Programming in GNU Radio: Maintainable and User-Friendly Algorithm Optimization with VOLK”, *WinnForum's SDR conference*, 2013.

[86] T. Rondeau, ”GNU Radio - signaalien ajoitus”. Verkkosivu. Haettu 04.06.2014 osoitteesta http://www.trondeau.com/storage/tutorial/gr_scheduler_overview.pdf.

[87] GNU Radio, ”GNU Radio sivuston tutoriaalit”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.org/redmine/projects/gnuradio/wiki/Tutorials.

[88] D. Kozel, ”SDR and GNU Radio Companion for Amateur Radio (tutoriaali)” . Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.pointview.com/data/files/17/16968/2601.pdf.

[89] GNU Radio, ”Ohjeita uusien GNU Radio moduulien toteuttamiseen”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules.

[90] Python Software Foundation, ”Python-ohjelmointikielen kotisivut”. Verkkosivu. Haettu 29.12.2014 osoitteesta https://www.python.org/.

[91] M. M. Kierulff, ”NumPy for CUDA”, Master's Thesis. Verkkosivu. Haettu 29.12.2014 osoitteesta https://cudanumpy.googlecode.com/svn/trunk/doc/CudaNumPy.pdf.

[92] Julkinen ohjelmistoprojekti, ”Numpy -projektin kotisivut (Avoin Python-ohjelmointikielen laajennos)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://www.numpy.org/.

[93] SWIG, ”SWIG -projektin kotisivut”. Verkkosivu. Haettu 29.12.2014 osoitteesta www.swig.org.

[94] SWIG, ”SWIG-projektin Github-arkisto”. Verkkosivu. Haettu 29.12.2014 osoitteesta https://github.com/swig/swig.

[95] GNU Radio, ”GNU Radio:n ohjesivusto (Doxygen)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.org/doc/doxygen/.

[96] GNU Radio & Corgan Labs, ”GNU Radio Live SDR -käyttöjärjestelmä”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioLiveDVD.

[97] Ettus Research LLC, ”GNU Radio:n asentamisohjeet Windows-käyttöjärjestelmälle”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://code.ettus.com/redmine/ettus/projects/uhd/wiki/GNURadio_Windows.

[98] Ettus Research LLC, ”USRP2-sarjan ohjelmistoradioiden sisäiset ajurit (USRP N210)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://code.ettus.com/redmine/ettus/projects/uhd/repository/revisions/master/show/fpga/usrp2.

[99] J.-O. Jeong, "Hybrid FPGA and GPP Implementation of IEEE 802.15.4 Physical Layer", Master's thesis, Blacksburg, Virginia: Virginia Polytechnic Institute and State University, 2012.

[100] F. A. Hamza, ”usrp_spectrum_sense.py -koodin esittely (ruby-forum)”. Verkkosivu. Haettu 29.12.2014 osoitteesta https://www.ruby-forum.com/topic/174437.

[101] T. Rondeau ja M. Ettus, ”Using GNU Radio to Explore the Consequences, Limits, and Behaviour of DSA systems”, IEEE DySPAN2014. Verkkosivu. Haettu 29.12.2014 osoitteesta http://gnuradio.squarespace.com/storage/examples/dyspan2014/gr.pdf.

[102] A. B. M. Bostaman, "Adjacent Channel Interference Mitigation Schemes for Software Defined Radio", Master's thesis, Tokio (Minato): Keio University - School of Integrated Design Engineering, 2008.

[103] P. Händel ja Z. Per, ”Receiver I/Q Imbalance: Tone Test, Sensitivity, Analysis, and The Universal Software Radio Peripheral”, *IEEE Transactions on instrumentation and measurement,* pp. 704-714, Maaliskuu 2009.

[104] Ettus Research LLC, ”USRP kalibraatiotyökalujen ohjesivu”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://files.ettus.com/manual/page_calibration.html.

[105] Agilent Technologies, ”Spectrum Analyzer Basics (5965-7920E)”. Verkkosivu. Haettu 29.12.2014 osoitteesta http://cp.literature.agilent.com/litweb/pdf/5965-7920E.pdf.

[106] H. Hassanieh, L. Shi, O. Abari, E. Hamed ja D. Katabi, ”Big Band: GHz-Wide Sensing and Decoding on Commodity Radios”, *INFOCOM, 2014 Proceedings IEEE*, Toronto, 2014.

USRP-malli	X300/X310	QR210	B200/B210	N200/N210	E100/E110	USRP1
Tietokoneyhteys	PCIe (erillinen ohjain) / 10 Gig. Eth	10 Gig. Eth.	USB 3.0	Gig. Eth.	Sisäinen / Gig. Eth.	USB 2.0
Tietokoneyhteyden kaistanleveys (MS/s 16b/8b)	200/200	60/60	61.44/61.44	25/50	8/16	8/*
Tytärkorttien paikkoja	2	0 (integroitu)	0 (integroitu)	2	2	4
ADC:n resoluutio (bittiä)	14	16	12	14	12	12
ADC:n näytteistysnopeus (MS/s)	200	120	61.44 (simplex)	100	64	64
DAC:n resoluutio (bittiä)	16	x	12	16	14	14
DAC:n näytteistysnopeus (MS/s)	800	x	61.44 (simplex)	400	128	128
MIMO yhteensopivuus	8x8	8x8	2x2	8x8	Ei	Kyllä
Sisäinen GPS-lukittu oskillaattori (lisäosa)	Kyllä	Kyllä	Ei	Kyllä	Kyllä	Ei
1PPS/Ref -sisääntulo	Kyllä	Kyllä	Kyllä	Kyllä	Kyllä	Ei

Liite 1: Ettus Research:n ohjelmistoradiojärjestelmien perusominaisuudet [63]

USRP-malli	USRP N200	USRP N210	USRP B210	USRP X300	USRP X310
FPGA:	Spartan 3A DSP XC3SD1800A	Spartan 3A DSP XC3SD3400A	Spartan 6 XC6SLX150	KINTEX 7 XC7K325T	KINTEX 7 XC7K410T
DSP48-ytimien määrä:	54 / 84 (64%)	96 / 126 (76%)	180	753 / 840 (90%)	1 453 / 1 540 (94%)
Logiikkasolujen määrä:	15 350 / 37 440 (41%)	31 690 / 53 712 (59%)	147 443	326 080	406 720
RAM-muistin määrä:	786K / 1 512K (52%)	1 542K / 2 268K (68%)	4 824K	16 020K	28 620K
Ulkoisen muistin määrä:	1Mt SRAM	1Mt SRAM	?	1Gt DDR3	1Gt DDR3
Piirin kellotaajuus:	100	100	100	250	250
RF kaistanleveys (Yhteensä):	50	50	56	640	640
Laitteen hinta:	1 340 €	1 520 €	975 €	3 450 €	4 240 €

Liite 2:USRP ohjelmistoradiokehitysalustojen parametrien vertailu. [63] [66]

LIITE 3
usrp_tb.py

```
#!/usr/bin/env python
# -*- coding: cp1252 -*-
#
# Copyright 2005,2007,2011 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#

from gnuradio import gr, eng_notation
from gnuradio import blocks
#from gnuradio import filter
from gnuradio import fft
from gnuradio import uhd
#from gnuradio import analog
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import sys
import struct
import threading
import time
import ConfigParser
import numpy as np
import os
#from multiprocessing import Pool

sys.stderr.write("Warning: this may have issues on some machines+Python version
combinations to seg fault due to the callback in bin_statitics.\n\n")

class ThreadClass(threading.Thread):
  def run(self):
    return

class tune(gr.feval_dd):
  """
  This class allows C++ code to callback into python.
  """
  def __init__(self, tb):
    gr.feval_dd.__init__(self)
    self.tb = tb

  def eval(self, ignore):
    """
    This method is called from gr.bin_statistics_f when it wants
    to change the center frequency.  This method tunes the front
    end to the new center frequency, and returns the new frequency
    as its result.
    """
    try:
      # We use this try block so that if something goes wrong
      # from here down, at least we'll have a prayer of knowing
      # what went wrong.  Without this, you get a very
      # mysterious:
      #
```

```
      #   terminate called after throwing an instance of
      #   'Swig::DirectorMethodException' Aborted
      #
      # message on stderr.
      new_freq = self.tb.set_next_freq()

      # wait until msgq is empty before continuing
      while(self.tb.msgq.full_p()):
        #print "msgq full, holding.."
        time.sleep(0.001)

      return new_freq

    except Exception, e:
      print "tune: Exception: ", e


class parse_msg(object):
  def __init__(self, msg):
    try:
      self.center_freq = msg.arg1()
      self.vlen = int(msg.arg2())
      assert(msg.length() == self.vlen * gr.sizeof_float)

      # FIXME consider using NumPy array
      t = msg.to_string()
      self.raw_data = t
      self.data = struct.unpack('%df' % (self.vlen,), t)

    except Exception, e:
      print "parse: Exception: ", e


class my_top_block(gr.top_block):

  def __init__(self, *initargs):
    gr.top_block.__init__(self)
    ################# Config ########################
    configParser = ConfigParser.RawConfigParser() #load configParser
    configParser.read(r'config.txt') #load the config -file
    self.device_addr = configParser.get('variables', 'addr')
    self.samp_rate = self.usrp_rate = configParser.get('variables', 'samp_rate')
    self.fft_size = configParser.get('variables', 'fft_size')
    self.freq_step_mult = configParser.get('variables', 'freq_step_mult')
    self.lo_offset = configParser.get('variables', 'lo_offset')
    self.tune_delay = configParser.get('variables', 'tune_delay')
    self.dwell_delay = configParser.get('variables', 'dwell_delay')
    self.gain = configParser.get('variables', 'gain')
    self.manualtune = configParser.getboolean('variables', 'manualtune')
    self.mimoavgcount = configParser.get('variables', 'mimo_avgcount')
    self.savespectrogram = configParser.getboolean('variables', 'savespectrogram')
    self.savePSD = configParser.getboolean('variables', 'savePSD')
    self.agcCalibration = configParser.getboolean('variables', 'agcCalibration')
    ################# Variables ######################
    self.start_time = 0
    self.stop_time = 0
    self.power = 0
    self.nowfreq = 0
    self.freqs = []
    self.freqsnum = 0
    self.freqsfile = ""
    self.tuneNow = True
    self.freqNow = 0
    self.ettusnumber = 0
    self.realtime = False
    self.vakiok = 0
    self.dBm = 0
    ##################################################
```

LIITE 3
usrp_tb.py

```
    usage = "usage: %prog [options] min_freq max_freq"
    parser = OptionParser(option_class=eng_option, usage=usage)
    #parser.add_option("-a", "--args", type="string", default=self.device_addr,
    #                  help="UHD device device address args [default=%default]")
    parser.add_option("-d", "--devnum", type="int", default=self.ettusnumber,
              help="Ettus device number [0 or 1] [default=%default]")
    parser.add_option("", "--spec", type="string", default=None,
                      help="Subdevice of UHD device where appropriate")
    parser.add_option("-A", "--antenna", type="string", default=None,
              help="select Rx Antenna where appropriate")
    parser.add_option("-s", "--samp-rate", type="eng_float", default=self.samp_rate,
              help="set sample rate [default=%default]")
    parser.add_option("-g", "--gain", type="eng_float", default=self.gain,
              help="set gain in dB (default is midpoint)")
    parser.add_option("", "--tune-delay", type="eng_float",
              default=self.tune_delay, metavar="SECS",
              help="time to delay (in seconds) after changing frequency
[default=%default]")
    parser.add_option("", "--dwell-delay", type="eng_float",
              default=self.dwell_delay, metavar="SECS",
              help="time to dwell (in seconds) at a given frequncy [default=%default]")
    parser.add_option("-F", "--fft-size", type="int", default=self.fft_size,
              help="specify number of FFT bins [default=%default]")
    # always try to use real-time scheduling
    #parser.add_option("", "--real-time", action="store_true", default=False,
    #                  help="Attempt to enable real-time scheduling")
    parser.add_option("", "--freq-step", type="eng_float", default=self.freq_step_mult,
              help="set frequency step multiplier (default 0.75)")
    parser.add_option("-l", "--lo-offset", type="eng_float", default=self.lo_offset,
              help="lo_offset in Hz [default=%default]")
    parser.add_option("", "--manualtune", action="store_true", default=False,
              help="enable manual tuning")
    parser.add_option("", "--MIMO", action="store_true", default=False,
              help="enable MIMO")
    (options, args) = parser.parse_args()
    if len(args) == 0:
       parser.print_help()
       sys.exit(1)
    elif len(args) == 1:
      self.min_freq = self.max_freq = eng_notation.str_to_num(args[0])
    elif len(args) > 1:
      self.min_freq = eng_notation.str_to_num(args[0])
      self.max_freq = eng_notation.str_to_num(args[1])
      if self.min_freq > self.max_freq: # swap them
        self.min_freq, self.max_freq = self.max_freq, self.min_freq

    #self.device_addr = options.args
    self.samp_rate = options.samp_rate
    self.fft_size = options.fft_size
    self.freq_step_mult = options.freq_step
    self.lo_offset = options.lo_offset
    self.tune_delay = options.tune_delay
    self.dwell_delay = options.dwell_delay
    self.gain = options.gain
    self.manualtune = options.manualtune
    self.MIMO = options.MIMO
    self.ettusnumber = options.devnum

    # Attempt to enable realtime scheduling
    r = gr.enable_realtime_scheduling()
    if r == gr.RT_OK:
      self.realtime = True; realtime = True
    else:
      self.realtime = False
      print "Note: failed to enable realtime scheduling"

    # Set usrp_source
```

LIITE 3
usrp_tb.py

```
#self.u = uhd.usrp_source(device_addr=self.device_addr,
#                          stream_args=uhd.stream_args('fc32'))
print "MIMO: ",self.MIMO
if (True and self.MIMO):
  self.tune_delay = 0.0
  self.samp_rate = 1000000

osdir = os.path.dirname(os.path.realpath(__file__))
elif self.ettusnumber == 0:
  self.u = uhd.usrp_source(device_addr="addr=10.7.6.17",
           stream_args=uhd.stream_args('fc32'))
  # Set the antenna
  self.u.set_antenna("TX/RX", 0)
  calibfile = "%s/calibration/agc0.npy" % osdir
else:
  self.u = uhd.usrp_source(device_addr="addr=10.7.6.18",
           stream_args=uhd.stream_args('fc32'))
  # Set the antenna
  self.u.set_antenna("RX2", 0)
  calibfile = "%s/calibration/agc1.npy" % osdir
#load agcCalibration
if (self.agcCalibration and os.path.isfile(calibfile)):
  self.agcValues = np.load(calibfile)

# Set the subdevice spec
if(options.spec):
  self.u.set_subdev_spec(options.spec, 0)

self.set_variables()

################## Blocks ########################
s2v = blocks.stream_to_vector(gr.sizeof_gr_complex, self.fft_size)

mywindow = fft.window.blackmanharris(self.fft_size)
ffter = fft.fft_vcc(self.fft_size, True, mywindow, True)
self.power = 0
for tap in mywindow:
  self.power += tap*tap

c2mag = blocks.complex_to_mag_squared(self.fft_size)
# FIXME the log10 primitive is dog slow -> do it in post-processing
#log = gr.nlog10_ff(10, self.fft_size,
#-20*math.log10(self.fft_size)-10*math.log10(self.power/self.fft_size))

# Set the freq_step to <= 75% of the actual data throughput.
# This allows us to discard the bins on both ends of the spectrum.
#self.freq_step = self.nearest_freq((0.70 * self.usrp_rate), self.channel_bandwidth)

self.msgq = gr.msg_queue(4096) #some buffer
self._tune_callback = tune(self) # hang on to this to keep it from being GC'd
stats = blocks.bin_statistics_f(self.fft_size, self.msgq,
             self._tune_callback, self.tune_delay,
             self.dwell_delay)

if options.gain is None:
  # if no gain was specified, use the mid-point in dB
  g = self.u.get_gain_range()
  options.gain = float(g.start()+g.stop())/2.0

self.set_gain(options.gain)
self.set_gain(0)

print "gain =", options.gain
print "lo_offset =", self.lo_offset

################## Connections ########################
# FIXME leave out the log10 until we speed it up
#self.connect(self.u, s2v, fft, c2mag, log, stats)
```

```
    self.connect(self.u, s2v, ffter, c2mag, stats)

  ################ Class functions ######################
  def __get__(self, obj, objtype):
    return self.val

  def __set__(self, obj, val):
    self.val = val

  def set_next_freq(self):
    try:
      target_freq = self.next_freq
      #manual tuning uses set_next_freq_single() calls
      #from main program to change the target freq
      if (self.manualtune):
        self.set_freq(target_freq)
      else:
        #MIMO loads freqs from a file
        if (self.MIMO):
          if (self.freqs):
            if (self.freqsnum >= len(self.freqs)):
              self.freqs = [float(line.strip()) for line in open(self.freqsfile)]
              self.freqs.sort()
              self.freqsnum = 0
            self.next_freq = self.freqs[self.freqsnum]
            while (self.freqsnum < len(self.freqs) and self.freqs[self.freqsnum] ==
self.next_freq):
              self.freqsnum +=1
          else: #if freqs not yet loaded
            self.freqs = [float(line.strip()) for line in open(self.freqsfile)]
            self.freqs.sort()
            self.freqsnum = 0
        #normal tuning just tunes in steps
        else:
          self.next_freq = self.next_freq + self.freq_step
          if self.next_freq > self.max_center_freq:
            self.next_freq = self.min_center_freq
        if not self.set_freq(target_freq):
          print "Failed to set frequency to", target_freq
          self.msgq.flush()
      return target_freq
    except Exception, e:
      print "set_next_freq Error: ", e
      self.msgq.flush()

  def set_freq(self, target_freq):
    try:
      #r = self.u.set_center_freq(target_freq)
      # loads calibration values and tunes the front-end gain
      if (self.agcCalibration):
        self.gainvalue = np.interp(target_freq, self.agcValues[1], self.agcValues[0])
        self.u.set_gain(self.gainvalue)
      # tunes the front-end frequency
      r = self.u.set_center_freq(uhd.tune_request(target_freq,
            rf_freq=(target_freq + self.lo_offset),rf_freq_policy=
            uhd.tune_request.POLICY_MANUAL))
      if r:
        self.freqNow = target_freq
        return True
      return False
    except Exception, e:
      print "set_freq Error: ", e
      self.msgq.flush()

  def set_gain(self, gain):
    self.u.set_gain(gain)

  def set_freqs(self,freqs):
```

LIITE 3
usrp_tb.py

```
    self.freqs = freqs
    self.next_freq = freqs[0]

  def set_variables(self):
    self.usrp_rate = self.samp_rate
    self.u.set_samp_rate(self.usrp_rate)
    self.dev_rate = self.u.get_samp_rate()
    self.freq_step = self.freq_step_mult * self.usrp_rate
    self.nsteps = int(round((self.max_freq - self.min_freq) / self.freq_step))
    self.min_center_freq = self.min_freq + self.freq_step/2
    self.max_center_freq = self.min_center_freq + (self.nsteps * self.freq_step)
    self.next_freq = self.min_center_freq
    print "samp_rate: ", self.samp_rate
    print "tune delay: ", self.tune_delay
    print "dwell delay: ", self.dwell_delay
    print "saveSpectrogram: ", self.savespectrogram
    print "savePSD: ", self.savePSD
    print "agcCalibration: ", self.agcCalibration

    self.tune_delay      =   max(0,   int(round(self.tune_delay   *   self.usrp_rate   /
self.fft_size)))  # in fft_frames
    self.dwell_delay    =    max(1,   int(round(self.dwell_delay   *   self.usrp_rate   /
self.fft_size))) # in fft_frames
```

LIITE 4
usrp_main.py

```
#!/usr/bin/env python
# -*- coding: cp1252 -*-
"""
Created on Mon Nov 3 11:50:00 2014
@author: Ville Saari
"""

# Main program for usrp_N210 spectrum sensing

import time
import os

### usrp_spektr3 libraries ###
# library for the top_block which controls the N210
from usrp_spektr3_tb import *
#from usrp_spektr3_tb_old import *
#from usrp_spektr3_tb_sim import *
# library for saving and loading configuration files
from usrp_spektr3_conf import *
# library for drawing pictures from the values
from usrp_spektr3_draw import *

import numpy as np

def makeFileFolders(daypath, ettusNumber):
  #make datapath for day if it does not exist
  if not os.path.exists(daypath):
    os.mkdir(daypath)
  if not os.path.exists("%s/kuvat" % daypath):
    os.mkdir("%s/kuvat" % daypath)
  if not os.path.exists("%s/data" % daypath):
    os.mkdir("%s/data" % daypath)

import subprocess
import re
def checkAntDir(): # function for getting the RFI-antenna direction
  return 999
  if (False): #does not work yet
    angle = 360
    callangle = '%d\r' % int(angle)
    call = "echo -en %s | nc auxant 4001 -w1 | cat -A" % callangle
    print call
    check = subprocess.check_output([call], shell=True)
    print "check: ", check
    #my @ohjaus=`/bin/echo -en 'M$angle\r' | nc auxant 4001 -w1 | cat -A`;
    time.sleep(1)

  done = False
  while not(done):
    value = subprocess.check_output(["echo 'C\r\n' | nc auxant 4001 -w1 | cat -
A"], shell=True)
    value = int(re.findall(r'\b\d+\b', value)[0])
    if (360-value <= 2 or value <= 2):
      value = 0; done = True;
    elif (88 <= value <= 92):
      value = 90; done = True;
    elif (178 <= value <= 182):
      value = 180; done = True;
    elif (268 <= value <= 272):
      value = 270; done = True;
    time.sleep(1)
```

```
  print "Angle: %d" % int(value)
  return int(value)

# Renames the given filename if such file
# (or subsequent filenames) already exist
# otherwise just returns the given filename
def measnumFindNext(filename):
  measnum = 0
  startpoint = filename.rfind('/')+1
  meascount = filename[startpoint:].count('_')
  endpoint = filename.rfind('.')
  if (meascount == 1 and os.path.isfile(filename)):
    filename = filename[:endpoint] + "_%d" % measnum + filename[endpoint:]
    measnum += 1
  while (os.path.isfile(filename)):
    startpoint = filename.rfind('_')
    filename = filename[:startpoint] + "_%d" % measnum + filename[endpoint:]
    measnum += 1
  return filename

# Renames the given filename with the given measnum
def measnumSet(filename, measnum):
  startpoint = filename.rfind('/')+1
  meascount = filename[startpoint:].count('_')
  endpoint = filename.rfind('.')
  if (meascount == 1):
    filename = filename[:endpoint] + "_%d" % measnum + filename[endpoint:]
  elif (meascount == 2):
    startpoint = filename.rfind('_')
    filename = filename[:startpoint] + "_%d" % measnum + filename[endpoint:]
  return filename

# Adds +1 to the given filenames measnum
def findmeasnum(filename):
  startpoint = filename.rfind('/')+1
  meascount = filename[startpoint:].count('_')
  endpoint = filename.rfind('.')
  startpoint = filename.rfind('_')+1
  if (meascount == 1):
    return -1
  elif (meascount == 2):
    measnum = int(filename[startpoint:endpoint])
  return measnum

# Checks if the jsonfile with same configs can be found
# returns the "measnum" to that jsonfile
def checkjsons(tb, jsonname):
  measnum = -1
  while (os.path.isfile(jsonname)):
    if (jsoncompare(tb, jsonname)):
      break
    else:
      measnum = findmeasnum(jsonname)+1
      jsonname = measnumSet(jsonname, measnum)
  return measnum


def main_loop(tb, draw_eng):
  try:
    #check if running in MIMO-mode
    MIMO = tb.MIMO
```

LIITE 4
usrp_main.py

```
    #check which Ettus N210 is running
    ettusNumber = tb.ettusnumber

    #set and make directories
    filedirectory = os.path.dirname(os.path.realpath(__file__))
    daypath    =    "%s/%s"    %    (filedirectory,    time.strftime("%Y_%m_%d",
time.gmtime()))
    dayNow = str(time.strftime("%d", time.gmtime()))
    hourNow = str(time.strftime("%H", time.gmtime()))
    makeFileFolders(daypath, ettusNumber)

    # Check RFI antenna direction
    antDir = checkAntDir()

    startupdelay = 500
    length = int((tb.nsteps+1)*tb.freq_step_mult*tb.fft_size)
    # load mimo freqs and set max files
    if (MIMO):
      measname = "%s/data/mimo%d_%d.npy" % (daypath, ettusNumber, antDir)
      measfile = '%s/data/mimo%d_%d.txt' % (daypath, ettusNumber, antDir)
      jsonname = "%s/data/mimo%d_%d.json" % (daypath, ettusNumber, antDir)
      freqsfile = '%s/taajuudet.txt' % daypath
      tb.set_freqsfile(freqsfile)
      mimoArray = np.zeros(((2048, length)), dtype = float32)
      mimoavgcount = int(tb.mimoavgcount)
    else: # normal mode
      measname = "%s/data/meas%d_%d.npy" % (daypath, ettusNumber, antDir)
      measfile = '%s/data/meas%d_%d.txt' % (daypath, ettusNumber, antDir)
      rawfile = "%s/data/raw%d_%d.npy" % (daypath, ettusNumber, antDir)
      jsonname = "%s/data/meas%d_%d.json" % (daypath, ettusNumber, antDir)

    measnum = -1
    # if a config file with same name is found
    tb.start_time = time.time()
    if (os.path.isfile(jsonname)):
      print "checking jsons"
      measnum = checkjsons(tb, jsonname)
      if (measnum >= 0):
        measname = measnumSet(measname, measnum)
        measfile = measnumSet(measfile, measnum)
        jsonname = measnumSet(jsonname, measnum)
        if not (MIMO):
          rawfile = measnumSet(rawfile, measnum)
      jsondata = loadjsondata(jsonname)
      tb.start_time = jsondata.starttime

    # load or make files for arrays ###
    if (os.path.isfile(measname)):
      measArray = np.load(measname)
    else:
      measArray = np.zeros((3, length), dtype = float32)
      measArray[2] += 100000 # format the min array
    measArray2 = np.zeros(length, dtype = float32)

    # calibration file folders
    if (tb.samp_rate > 1000000):
      loaddir = "calibration/%sM/" % int(tb.samp_rate/1000000)
    else:
      loaddir = "calibration/%sk/" % int(tb.samp_rate/1000)

    loadname = "%s/zeroes%s.npy" % (loaddir, tb.fft_size)
```

```
    if (os.path.isfile(loadname)):
      arrayZeros = np.load(loadname)
      zeroing = True
    else:
      print "no zeroes -file!"
      zeroing = False

    # start the top_block ###
    tb.start()

    # load rest of the variables used in main program
    freqclock = timekeeper = time.time()
    samplerate = tb.samp_rate
    fftsize = tb.fft_size; halfsize = fftsize/2
    freqstepmult = tb.freq_step_mult
    dpstep = int(fftsize*freqstepmult)
    nsteps = tb.nsteps
    mincenterfreq = tb.min_center_freq
    maxcenterfreq = tb.max_center_freq
    samplestep = samplerate*freqstepmult
    fftstep = samplestep/fftsize
    vakiok = 20*np.log10(fftsize)+10*np.log10(tb.power/fftsize)
    tb.vakiok = vakiok
    tb.dBm=  97.548  -  20*log10(fftsize/128)  +  10*log10(samplerate/10000000)  +
vakiok
    stepNow = 0
    draw = save = 0
    arrstart = int((1-freqstepmult)/2*fftsize)
    arrstop = arrstart+dpstep
    threshold = 1/(pow(10, (65-vakiok)/10))

    mimoSave = False
    mimoSaveCount = 0
    sweeptimeEval = (tb.tune_delay+tb.dwell_delay)*nsteps*fftsize/samplerate

    startup = antennaRotated = False
    while (True):
      # Get the next message sent from the C++ code (blocking call)
      # It contains the center frequency and the mag squared of the fft
      m = parse_msg(tb.msgq.delete_head())

      if (startup and mincenterfreq >= m.center_freq <= maxcenterfreq):
        # #print center freq so we know that something is happening
        #print m.center_freq

        # arrayNow selects the middle 75% of the received fft-bins
        arrayNow = np.array(m.data[arrstart:arrstop])

        if not(MIMO):
          # Calculate which fft-bins we are looking at
          stepNow = int((m.center_freq-mincenterfreq)/(samplestep))

          # start points to the corresponding starting bin in measArray
          start = stepNow*dpstep

          # do the calibration
          if (zeroing):
            arrayNow -= arrayZeros[start:(start+dpstep)]
          #if (calibration):
          #    arrayNow *= maxCalibration[start:(start+dpstep)]
```

```
          # measArray2 is max of the filtered part of the received data
          measArray2[start:(start+dpstep)]                                    =
np.maximum(measArray2[start:(start+dpstep)], arrayNow)

          # measArray is max of the filtered part of the received data
          measArray[0,start:(start+dpstep)]=np.maximum(measArray[0,
start:(start+dpstep)], arrayNow)
          measCount = measArray[1, -1]
          measArray[1, start:(start+dpstep)] += arrayNow
          measArray[2,start:(start+dpstep)]=np.minimum(measArray[2,
start:(start+dpstep)], arrayNow)

        if (True):
          # check the largest signal we received
          if (np.max(arrayNow) > threshold):
            ind = arrayNow.argsort() #sort array
            i = -1 #last things first..
            with open(measfile, 'a') as f:
              while (i > -1*len(arrayNow) and arrayNow[ind[i]] > threshold):
                valueNow = 10*np.log10(arrayNow[ind[i]])-vakiok
                taajuusSpot = m.center_freq + fftstep*(ind[i]-halfsize)
                f.write(str(taajuusSpot)+","+str(valueNow)\
                      +","+str(time.strftime("%H:%M:%S", time.gmtime()))\
                      +","+str(time.time())+"\n")
                i -=1
            if (False):
              if not(MIMO):
                # if not MIMO then save some freqs for controlling the MIMO-N210
                if (time.time()-freqclock > 120):
                  with open('%s/taajuudet.txt' % daypath, 'w') as f:
                    f.write(str(m.center_freq)+"\n")
                    freqclock = time.time()
                else:
                  with open('%s/taajuudet.txt' % daypath, 'a') as f:
                    f.write(str(m.center_freq)+"\n")

      if (startup):
        #print "stepNow %d / %d" % (stepNow, nsteps) #debug
        #print "cfreq %f / %f" % (m.center_freq, maxcenterfreq) #debug
        if (MIMO):
          save +=1
          mimoArray[mimoSaveCount] += arrayNow
          if (save >= mimoavgcount):
            if not (mimoSave):
              maxNow = np.max(mimoArray[mimoSaveCount])/mimoavgcount
              if (maxNow > threshold):
                #print "measuring.."
                mimoSave = True
              else:
                mimoArray[mimoSaveCount] = 0

            else: #if (mimoSave):
              mimoSaveCount += 1
              if (mimoSaveCount >= 2048):
                mimoArray /= mimoavgcount
                np.save(filename, mimoArray)
                tb.stoptime = time.gmtime()
                jsondumpdatas(tb, jsonname)
                measnum += 1
                measname = measnumSet(measname, measnum)
                measfile = measnumSet(measfile, measnum)
```

```
                jsonname = measnumSet(jsonname, measnum)
                mimoSave = False
                startup = False #give some time for saving
                mimoArray = np.zeros(((2048, length)), dtype=float32)
                mimoSaveCount = 0
            save = 0

        elif (stepNow == nsteps): #### NOT MIMO #####
            save +=1
            draw +=1
            measArray[1, -1] = measCount + 1
            # for every n: send measured samples to draw_eng
            if (tb.savespectrogram and draw >= 1):
              measArray2 /= draw
              draw_eng.memArray(measArray2, jsonPassdata(tb), rawfile)
              measArray2 = np.zeros(length)
              draw = 0

            if not(antDir == checkAntDir()):
              antennaRotated = True

          # if the hour changes or the RFI-antenna is rotated
          # -> save files + change output files
          if (antennaRotated or(not(hourNow == time.strftime("%H", time.gmtime()))
and tb.savespectrogram)):

            hourNow = str(time.strftime("%H", time.gmtime()))
            antDir = checkAntDir()
            antennaRotated = False
            np.save(measname, measArray)
            jsondumpdatas(tb, jsonname)
            tb.start_time = time.time()
            if (tb.savespectrogram and not(MIMO)):
              draw_eng.saveArray = True
              draw_eng.memArray(measArray2, jsonPassdata(tb), rawfile)
              draw_eng.clearRawArray(jsonPassdata(tb))
            if not(str(time.strftime("%d", time.gmtime()) == dayNow)):
              dayNow = str(time.strftime("%d", time.gmtime()))
              daypath=   "%s/%s"   %   (filedirectory,   time.strftime("%Y_%m_%d",
time.gmtime()))
              makeFileFolders(daypath)
            measnum += 1
            measname = measnumSet(measname, measnum)
            measfile = measnumSet(measfile, measnum)
            jsonname = measnumSet(jsonname, measnum)
            if not (MIMO):
              rawfile = measnumSet(rawfile, measnum)
              measArray = np.zeros((3, length), dtype = float32)
              measArray[2] += 100000

            save = 0
            draw = 0
            startup = False # just give the process some time to save

        # if not(startup)
        elif (save >= startupdelay):
          startup = True
          tb.msgq.flush()
          #print "started.."
          save = 0
        else:
```

```
            tb.msgq.flush()
            save +=1
            if not(MIMO):
              tb.next_freq = mincenterfreq

        #--While(True): ends---

    # if the program is shutdown by ctrl C+X -> stop USRP, save files and quit
    except KeyboardInterrupt:
        print "Interrupted"
        # stop the top_block
        tb.stop()
        print 'top block stopped'
        if (tb.savespectrogram and not(MIMO)):
          draw_eng.saveArray = True
          draw_eng.memArray(measArray2, jsonPassdata(tb), rawfile)
        np.save(measname, measArray)
        jsondumpdatas(tb, jsonname)
        return 0
    #except (RuntimeError, TypeError, NameError):

if __name__ == '__main__':
    t = ThreadClass()
    t.start()
    tb = my_top_block()
    draw_eng = draw_engine()
    try:
        main_loop(tb, draw_eng)
    except [[KeyboardInterrupt]]:
        pass
    t.join()
    #print('Main Terminating...')
```

LIITE 5
usrp_conf.py

```
#!/usr/bin/env python
# -*- coding: cp1252 -*-
"""
Created on Mon Nov 14 8:23:11 2014
@author: Ville Saari
"""

import json
import codecs
import os
import time

#Function for saving configurations
def jsondumpdatas(tb, filename):
  output_file = codecs.open(filename, "w", encoding="utf-8")
  j = {"samp-rate": tb.samp_rate,
     "gain": tb.gain,
     "tune-delay": tb.tune_delay,
     "dwell-delay": tb.dwell_delay,
     "fft-size": tb.fft_size,
     "start_frequency": tb.min_center_freq\
       -tb.samp_rate*tb.freq_step_mult/2,
     "stop_frequency": tb.max_center_freq\
       +tb.samp_rate*tb.freq_step_mult/2,
     "min_center_frequency": tb.min_center_freq,
     "max_center_frequency": tb.max_center_freq,
     "nsteps": tb.nsteps,
     "start_time": tb.start_time,
     "stop_time": time.time(),
     "freq_step_mult": tb.freq_step_mult,
     "power": tb.power,
     "filelen": 0, #len(tb.m2freqs())
     "mimo_avgcount": tb.mimoavgcount,
     "vakiok": tb.vakiok,
     "dBm": tb.dBm
     }
  json.dump(j, output_file, indent=4, sort_keys=True, ensure_ascii=False)

def jsonPassdata(tb):
  j = {"samp-rate": tb.samp_rate,
     "gain": tb.gain,
     "tune-delay": tb.tune_delay,
     "dwell-delay": tb.dwell_delay,
     "fft-size": tb.fft_size,
     "start_frequency": tb.min_center_freq\
       -tb.samp_rate*tb.freq_step_mult/2,
     "stop_frequency": tb.max_center_freq\
       +tb.samp_rate*tb.freq_step_mult/2,
     "min_center_frequency": tb.min_center_freq,
     "max_center_frequency": tb.max_center_freq,
     "nsteps": tb.nsteps,
     "start_time": tb.start_time,
     "stop_time": tb.stop_time,
     "freq_step_mult": tb.freq_step_mult,
     "power": tb.power,
     "filelen": 0,
     "mimo_avgcount": tb.mimoavgcount,
     "vakiok": tb.vakiok,
     "dBm": tb.dBm
     }
  return j
```

LIITE 5
usrp_conf.py

```
def loadsettings(values):
  a = Datasettings(values["samp-rate"],
    values["gain"], values["tune-delay"],
    values["dwell-delay"],values["fft-size"],
    values["start_frequency"],
    values["stop_frequency"],
    values["min_center_frequency"],
    values["max_center_frequency"],
    values["nsteps"], values["start_time"],
    values["stop_time"], values["freq_step_mult"],
    values["power"], values["filelen"], values["mimo_avgcount"],
    values["vakiok"], values["dBm"])
  return a

def loadjsondata(filename):
  if not (filename[-4:] == "json"):
    if (filename[-3:] == "npy"):
      filename = filename[-3:]+"json"

  if (os.path.isfile(filename)):
    with open (filename) as f:
      arvotvalues = json.loads(f.read().decode("utf-8-sig"))
      a = loadsettings(arvotvalues)
      return a
  else:
    return False

def jsoncompare(tb, filename):
  jsonlast = loadjsondata(filename)
  if (jsonlast.sr == tb.samp_rate)\
  and (jsonlast.g == tb.gain)\
  and (jsonlast.td == tb.tune_delay)\
  and (jsonlast.dd == tb.dwell_delay)\
  and (jsonlast.fft == tb.fft_size)\
  and (jsonlast.startf == tb.min_center_freq\
     -tb.samp_rate*tb.freq_step_mult/2)\
  and (jsonlast.stopf == tb.max_center_freq\
     +tb.samp_rate*tb.freq_step_mult/2)\
  and (jsonlast.mincf == tb.min_center_freq)\
  and (jsonlast.maxcf == tb.max_center_freq)\
  and (jsonlast.nsteps == tb.nsteps)\
  and (jsonlast.fsmult == tb.freq_step_mult)\
  and (jsonlast.power == tb.power):
    return True
  else:
    return False

class Datasettings:
  def __init__(self, samp_rate, gain, tune_delay,
         dwell_delay, fft_size, start_frequency,
         stop_frequency, min_center_frequency,
         max_center_frequency, nsteps, start_time,
         stop_time, freq_step_mult, power, flen,
         mavgcount, vakiok, dBm):
    self.sr = samp_rate
    self.g = gain
    self.td = tune_delay
    self.dd = dwell_delay
    self.fft = fft_size
    self.startf = start_frequency
```

```
self.stopf = stop_frequency
self.mincf = min_center_frequency
self.maxcf = max_center_frequency
self.nsteps = nsteps
self.starttime = start_time
self.stoptime = stop_time
self.fsmult = freq_step_mult
self.power = power
self.flen = flen
self.mavgcount = mavgcount
self.vakiok = vakiok
self.dBm = dBm
```

LIITE 6
usrp_draw.py

```
#!/usr/bin/env python
# -*- coding: cp1252 -*-
"""
Created on Mon Nov 5 12:12:02 2014
@author: Ville Saari
"""

# usrp_spektr3 library for drawing functions

import matplotlib as mpl
mpl.use('Agg') #don't draw the images on screen
import matplotlib.pyplot as plt #read the help here:
http://matplotlib.org/users/pyplot_tutorial.html
import matplotlib.image as mpimg
import numpy as np
from pylab import *
import os
import time
import gc
#from threading import Thread
# __rdtsc() ?

### usrp_spektr3 libraries ###
from usrp_spektr3_conf import *

class draw_engine():

  def __init__(self):
    #Thread.__init__(self)
    ################# Variables ####################
    self.start_time = 0
    self.img = None
    self.jetmap = plt.get_cmap('jet')
    self.jetmap_r = plt.get_cmap('jet_r')
    self.saveArray = False
    self.saveMIMOArray = False
    self.savedMIMO = False
    self.mimoTimer = None
    self.vakiok = None
    self.frequencies = None
    self.raw = None
    self.rawMIMO = None
    self.yposMIMO = 0
    self.gain = None
    self.dBm = None
    self.sweepTime = None
    self.ypixelcount = 0
    ################################################

  # formatting function for the class
  def formatNones(self, jsondata):
    if (type(jsondata) is dict or isinstance(jsondata, str)): #change datatype
      jsondata = loadsettings(jsondata)
    #self.vakiok = 20*math.log10(jsondata['fft-
size'])+10*math.log10(jsondata['power']/jsondata['fft-size'])
    self.vakiok =
20*math.log10(jsondata.fft)+10*math.log10(jsondata.power/jsondata.fft)
    #self.gain = jsondata['gain']
    self.gain = jsondata.g
    #self.dBm = 97.548 - 20*log10(jsondata['fft-size']/128) +
10*log10(jsondata['samp-rate']/10000000)
```

```
    self.dBm = 97.548 - 20*log10(jsondata.fft/128)# +
10*log10(jsondata.sr/10000000)?
    self.dBm += self.vakiok
    self.dBm -= 20 #FIXME ?
    self.sweepTime = jsondata.nsteps*((jsondata.fft/jsondata.sr\
      *(jsondata.dd+jsondata.td)))

  # getters & setters
  def __get__(self, obj, objtype):
    return self.val

  def __set__(self, obj, val):
    self.val = val

  #"Max hold" function, which compresses the array
  #and makes the data easier to draw
  def drawResize(self, array, size):
    #divide the array in N parts and keep
    #the max valued bin in N
    amplitudes = np.zeros(size)
    if (array.ndim == 1): #one dimensional
      if (len(array) < size):
        size = len(array)
      perbin = int(len(array)/size)
      for i in range (0, size):
        idx = (array[i*perbin:i*perbin+perbin]).argmax()
        amplitudes[i] = array[i*perbin+idx]
      return amplitudes
    elif (array.ndim == 2): #two dimensional
      if (len(array[0]) < size):
        size = len(array[0])
      frequencies = np.zeros(size)
      perbin = int(len(array[0])/size)
      for i in range (0, size):
        idx = (array[0, i*perbin:i*perbin+perbin]).argmax()
        amplitudes[i] = array[0, i*perbin+idx]
        frequencies[i] = array[1, i*perbin+idx]
      myArray = np.vstack((amplitudes, frequencies))
      return myArray
    print "drawResize: not a correct array?"
    return False

  # just a format function for the spectrogram array
  def clearRawArray(self, jsondata):
    self.formatNones(jsondata)
    if (self.sweepTime > 1.5):
      self.ypixelcount = int(3600/self.sweepTime)
    else:
      self.ypixelcount = 2000
    self.raw = np.zeros(((self.ypixelcount, 4096)), dtype=float32)

  # array for holding the spectrogram values
  def memArray(self, amplitudes, jsondata, filename):
    self.formatNones(jsondata)

    if (self.raw == None):
      if (os.path.isfile(filename)):
        self.raw = np.load(filename)
      else:
        self.clearRawArray(jsondata)
```

```
    # Compress the frequency axis
    myArray = self.drawResize(amplitudes, 4096)

    # Calculate the timebin and set binvalues
    secondsNow = int(time.strftime("%M", time.gmtime()))*60\
      +int(time.strftime("%S", time.gmtime()))
    yposNow = int(round(secondsNow/3600.0*self.ypixelcount))
    #print "yposNow: %d" % yposNow
    if (yposNow < len(self.raw)):
      if (np.max(self.raw[yposNow]) == 0):
        self.raw[yposNow] = myArray
      elif (yposNow+1) < len(self.raw):
        yposNow +=1
        self.raw[yposNow] = myArray

    if (self.saveArray):
      np.save(filename, self.raw)
      self.saveArray = False
      #print "raw data saved"

  # dBm -> dBW/m2/Hz (copy from old RFI-drawing script)
  def dBwperm2(self, array, minf, maxf):
    alus1 = len(array)
    sa_conv = np.zeros(alus1); sfu1 = np.zeros(alus1)
    K_factor = np.zeros(alus1); G_amp = np.zeros(alus1)
    A_cab = np.zeros(alus1); G_antenna = np.zeros(alus1);
    E_total = np.zeros(alus1);
    freq = np.linspace(minf,maxf,alus1) #frequency vector
    # Antenna K-factor
    K_factor=-0.00000000294*freq*freq*freq+0.0000167*freq*freq-0.0202*freq+26.1
    # Gain of the amplifier (dB)
    G_amp=-0.0000000058*freq*freq*freq+0.000019*freq*freq-0.013*freq+27
    # Attenuation of the cable (dB)
    A_cab=-0.0000000012*freq*freq*freq+0.0000095*freq*freq-0.021*freq+0.41
    # Antenna gain
    G_antenna=0.0000000046*freq*freq*freq-0.000027*freq*freq+0.045*freq-12
    # Spectrum analyzer, conversion to dBm to dBuV
    sa_conv=array+107-G_amp-A_cab
    # Total field strength
    E_total=sa_conv+K_factor #<- should this be used somewhere?
    # Power lux density (dB.W.m^-2.Hz^-1)
    sfu1=array-G_amp-A_cab-G_antenna-10*log(3)/log(10)+20*log(freq)/log(10)-
95.54
    return sfu1

  # function for compressing the image (Max hold)
  def compressImage(self, im, x, y):
    perbin = int(len(im))/y

    for xi in range (0, x):
      for yi in range (0, y):
        im[xi, yi] = np.nanmax(im[i:i+perbin], axis = 0)
        im[i] /= perbin
      im = im[:size]

    perbin = int(len(im[0]))/x
    #for i in range ()
    for i in range (0, x):
      im[i] = np.nanmax(im[i:i+perbin], axis = 0)
      im[i] /= perbin
    im = im[:size]
```

```
    return im

  # function that loads the maxavgmin array
  # the avg-array's last value [1,-1] is the count of avgs
  def loadmaxavgmin(self, filename):
    myArray = np.load(filename)
    myArray[1] /= (myArray[1,-1])
    myArray[1,-1] = 0
    myArray = 10*np.log10(myArray) - self.vakiok
    return myArray

  # filtering function for drawing the maxavgmin -array
  def drawFilter(self, myArray):
    filtArray = np.copy(myArray[1])
    filtArray += self.dBm
    filtArray[~np.isfinite(filtArray)] = 0
    filtArray[np.abs(myArray[2]-myArray[0]) > 40] = 0
    myArray -= filtArray
    del filtArray
    return myArray

  # Function for drawing max-, avg- and min amplitudes (+PSD)
  def saveMeasArray(self, datafile, jsondata, filename, psd):
    np.seterr(divide='ignore') #ignore divide by zero
    plt.ioff() # no GUI -> don't draw on screen
    self.formatNones(jsondata)

    amplitudes = self.loadmaxavgmin(datafile)
    if (True):
      amplitudes = self.drawFilter(amplitudes)

    self.frequencies = np.linspace(jsondata.startf, jsondata.stopf,
len(amplitudes[0]))
    self.frequencies /= 1000000000

    maxArray = self.drawResize(np.vstack((amplitudes[0], self.frequencies)),
20000)
    avgArray = self.drawResize(np.vstack((amplitudes[1], self.frequencies)),
20000)
    minArray = self.drawResize(np.vstack((amplitudes[2], self.frequencies)),
20000)

    if (psd):
      maxArray[0] = self.dBwperm2(maxArray[0], jsondata.startf/1000000,
jsondata.stopf/1000000)
      avgArray[0] = self.dBwperm2(avgArray[0], jsondata.startf/1000000,
jsondata.stopf/1000000)
      minArray[0] = self.dBwperm2(minArray[0], jsondata.startf/1000000,
jsondata.stopf/1000000)

    f = figure(figsize=(18,10))
    plt.plot(maxArray[1], maxArray[0], 'r', linewidth=0.3, label="maximum")
    plt.plot(avgArray[1], avgArray[0], 'b', linewidth=0.3, label="average")
    plt.plot(minArray[1], minArray[0], 'g', linewidth=0.3, label="minimum")
    plt.legend()

    # axis tics - for y-axis
    maxAmplitude = np.nanmax(maxArray[0]+5)
    minAmplitude = np.nanmin(minArray[0])
    yaxismax = maxAmplitude-maxAmplitude%10
    yaxismin = minAmplitude-minAmplitude%10
```

```
    tick_locs = np.linspace(yaxismin, yaxismax, (yaxismax-yaxismin)/10+1)
    plt.yticks(tick_locs)
    # for x-axis
    xaxismin = round(np.min(self.frequencies),1)
    xaxismax = round(np.max(self.frequencies),1)
    plt.axis([xaxismin, xaxismax, minAmplitude, maxAmplitude])
    tick_locs = np.linspace(xaxismin, xaxismax, (xaxismax-xaxismin)*10+1)
    plt.xticks(tick_locs)

    starttime = time.strftime("%H:%M:%S", time.gmtime(jsondata.starttime))
    stoptime = time.strftime("%H:%M:%S", time.gmtime(jsondata.stoptime))
    date = time.strftime("%d/%m/%Y", time.localtime(jsondata.starttime))
    title = "Date: %s      Time: %s - %s" % (date, str(starttime),str(stoptime))
    plt.title(title)
    plt.xlabel("Frequency [GHz]")
    if (psd):
      plt.ylabel("PSD [dBW/m2/Hz]")
    else:
      plt.ylabel("Amplitude [dBm]")

    plt.grid()
    plt.savefig((filename), dpi=round(294), bbox_inches='tight')
    plt.clf()
    plt.close()
    plt.close(f)
    del maxArray
    del avgArray
    del minArray
    del amplitudes

  # function for drawing the spectrogram images
  def saveSpectrogram(self, datafile, jsondata, filename, mimo):
    np.seterr(divide='ignore') #ignore divide by zero
    plt.ioff() # no GUI -> don't draw on screen
    self.formatNones(jsondata)

    imgdata = np.load(datafile) # load data
    if (False): # saveSpectrogram seems to leak(?) memory so this might be
useful
      xsize = len(imgdata[0])
      ysize = len(imgdata)
      del imgdata
      imgdata = np.memmap(datafile, dtype=np.float32, mode='r', shape=(ysize,
xsize))


    # values to dBm + shift numbers to positive values by +self.dBm
    imgdata = 10*np.log10(imgdata) - self.vakiok
    #imgdata[imgdata<-97] = -1*self.dBm #compress low values to "zero"

    # scale numbers to range 0.0 - 1.0 -> we assume that
    # received signals are in range = -10 ... -1*self.dBm
    imgdata += self.dBm
    imgdata /= (-10+self.dBm)
    imgdata = self.jetmap(imgdata)

    f = figure(figsize=(16,9))
    plt.clf()

    starthour = int(time.strftime("%H", time.localtime(jsondata.starttime)))
```

```
    if (mimo):
      # Set the x-axis ticks
      xaxismin = round((centerfreq-jsondata.sr*jsondata.fsmult/2),1)
      xaxismax = round((centerfreq+jsondata.sr*jsondata.fsmult/2),1)
      xaxismin /= 1000000000
      xaxismax /= 1000000000

      tick_locs = np.linspace(0, len(amplitudes[0]), 15)
      ticks_axis = np.linspace(xaxismin, xaxismax, 15)
      tick_lbls = []
      for lbl in ticks_axis:
        tick_lbls.append("%0.4f" % lbl)
      plt.xticks(tick_locs, tick_lbls)

      # Set the y-axis ticks
      tick_locs = np.linspace(0,2048,15)
      tick_lbls = []
      for i in range(0,15):
        tick_lbls.append("%0.1f" % ((timespan/14.0)*i))
      plt.yticks(tick_locs, tick_lbls)

      # Set the image and axis titles
      timestamp = "%02d:%02d:%02d" % (hourvalue, minvalue, secvalue)
      title = "MIMO starting at ~%s" % timestamp

    else:
      # Set the x-axis ticks
      xaxismin = round(np.min((jsondata.startf/1000000000.0)),1)
      xaxismax = round(np.max((jsondata.stopf/1000000000.0)),1)
      tick_locs = np.linspace(0, len(imgdata[0])-1, (xaxismax-xaxismin)*10+1)
      tick_lbls = np.linspace(xaxismin, xaxismax, (xaxismax-xaxismin)*10+1)
      plt.xticks(tick_locs, tick_lbls)

      # Set the y-axis ticks
      tick_locs = np.linspace(0,len(imgdata)-1,15)
      minute = np.linspace(0,60,15)
      tick_lbls = []
      for i in range(0,14):
        tick_lbls.append("%02d:%02d" % (starthour, int(minute[i])))
      tick_lbls.append("%02d:00" % (starthour+1))
      plt.yticks(tick_locs, tick_lbls)

      # Set the image and axis titles
      date = time.strftime("%d/%m/%Y", time.localtime(jsondata.starttime))
      title = "Date: %s" % (date)

    # Draw the image, titles and labels
    im = plt.imshow(imgdata, interpolation='none')#, extent=[xaxismin, xaxismax,
1.0, 0.0])
    plt.title(title)
    plt.xlabel("Frequency [GHz]")
    plt.ylabel("Time")

    # Set the colorbar
    if (mimo):
      cb = plt.colorbar(im, fraction=0.015, pad=0.015)
    else:
      cb = plt.colorbar(im, fraction=0.025, pad=0.02)
    cmax = -10  #colorbar max
    cmin = -1*self.dBm
    plt.clim(cmin, cmax)
```

```
cb.set_label('Amplitude [dBm]')

# Save figure
#mpimg.imsave(filename, imgdata)
my_dpi = 308
plt.savefig((filename), dpi=my_dpi, bbox_inches='tight')

del imgdata
del im
plt.clf()
plt.cla()
plt.close()
plt.close(f)
close()
gc.collect()
```

LIITE 7
usrp_imag.py

```
#!/usr/bin/env python
# -*- coding: cp1252 -*-
"""
Created on Mon Nov 10 17:16:02 2014
@author: Ville Saari
"""
# a helper Program for drawing the spectrogram and
# spectrum images from the collected data

import os
import time
import glob
from optparse import OptionParser
from gnuradio.eng_option import eng_option
# library for saving and loading configuration files
from usrp_spektr3_conf import *
# library for drawing pictures from the values
from usrp_spektr3_draw import *

def main_loop():
  try:
    # set default directories
    filedirectory = os.path.dirname(os.path.realpath(__file__))
    daypath = "%s/%s" % (filedirectory, time.strftime("%Y_%m_%d",
time.gmtime()))
    dayNow = time.strftime("%d", time.gmtime())
    hourNow = time.strftime("%H", time.gmtime())

    # command line args
    usage = "usage: %prog [options]"
    parser = OptionParser(option_class=eng_option, usage=usage)
    parser.add_option("-f", "--filedirectory", type="string", default=daypath,
              help="filedirectory for files to operate on")
    parser.add_option("--cont", action="store_true",  default=False,
              help="enable continuous mode")
    parser.add_option("--dbm", action="store_true",  default=False,
              help="print dBm charts too")
    (options, args) = parser.parse_args()
    if (len(options.filedirectory) == 10):
      daypath = "%s/%s" % (filedirectory, options.filedirectory)
    daypath = options.filedirectory
    runCont = options.cont
    printDBM = options.dbm

    # set rest of the directories
    imagepath = "%s/kuvat" % (daypath)
    datapath = "%s/data" % (daypath)

    jsondata = False
    draw_eng = draw_engine()

    print "daypath: %s" % daypath
    while (True):
      # Check the filepath for raw-files &
      # save images for files that have not yet been saved
      filelist = sorted(glob.glob(os.path.join(datapath, 'raw*.npy')))
      for filename in filelist:
        #if the main program is not saving (giving some time to save)
        if (time.time()-os.path.getmtime(filename) > 1.5):
          ettusnumber = int(filename[(filename.find("raw")+3)])
          jsonfile = filename.replace("raw", "meas")
```

```
            jsonfile = jsonfile.replace(".npy", ".json")
            if not(os.path.isfile(jsonfile)):
              print "jsonfile not found!"
              break
            jsondata = loadjsondata(jsonfile)
            print "loaded: %s" % filename
            print "loaded: %s" % jsonfile
            startpoint = filename.rfind('_')+1
            endpoint = filename.rfind('.')
            measnumber = int(filename[startpoint:endpoint])
            file1 = "%s/spectrogram%d_%d.png" % (imagepath, ettusnumber,
measnumber)
            if not(os.path.isfile(file1)):
              draw_eng.saveSpectrogram(filename, jsondata, file1, False)
              print "saved %s" % (file1)

        filelist = sorted(glob.glob(os.path.join(datapath, 'meas*.npy')))
        for filename in filelist:
          #if the main program is not saving (giving some time to save)
          if (time.time()-os.path.getmtime(filename) > 1.5):
            ettusnumber = int(filename[(filename.find("meas")+4)])
            jsonfile = filename.replace(".npy", ".json")
            jsondata = loadjsondata(jsonfile)
            print "loaded: %s" % filename

            if (printDBM): #don't draw dBm image
              measpng = "%s/meas%s.png" % (imagepath,filename[-12:-4])
              if not(os.path.isfile(measpng)):
                draw_eng.saveMeasArray(filename, jsondata, measpng, False)
                print "saved %s. time: %s" % (measpng, time.strftime("%H:%M:%S",
time.gmtime()))

            measpngsfu = "%s/sfumeas%s.png" % (imagepath,filename[-12:-4])
            if not(os.path.isfile(measpngsfu)):
              draw_eng.saveMeasArray(filename, jsondata, measpngsfu, True)
              print "saved %s. time: %s" % (measpngsfu, time.strftime("%H:%M:%S",
time.gmtime()))

        if not(runCont):
          break
        elif (not(time.strftime("%H") == hourNow)):
          hourNow = time.strftime("%H")
          if not(time.strftime("%d") == dayNow):
            dayNow = gmtime.strftime("%d")
            daypath = "%s/%s" % (filedirectory, time.strftime("%Y_%m_%d",
time.gmtime()))
          imagepath = "%s/kuvat" % (daypath)
          datapath = "%s/data" % (daypath)
          jsondata = False

        print "waiting.."
        time.sleep(300)
    except KeyboardInterrupt:
      print "Stopped"
      return 0

if __name__ == '__main__':
    try:
      main_loop()
    except [[KeyboardInterrupt]]:
      pass
```